Prescriptive Process Models / Software Development Life Cycle

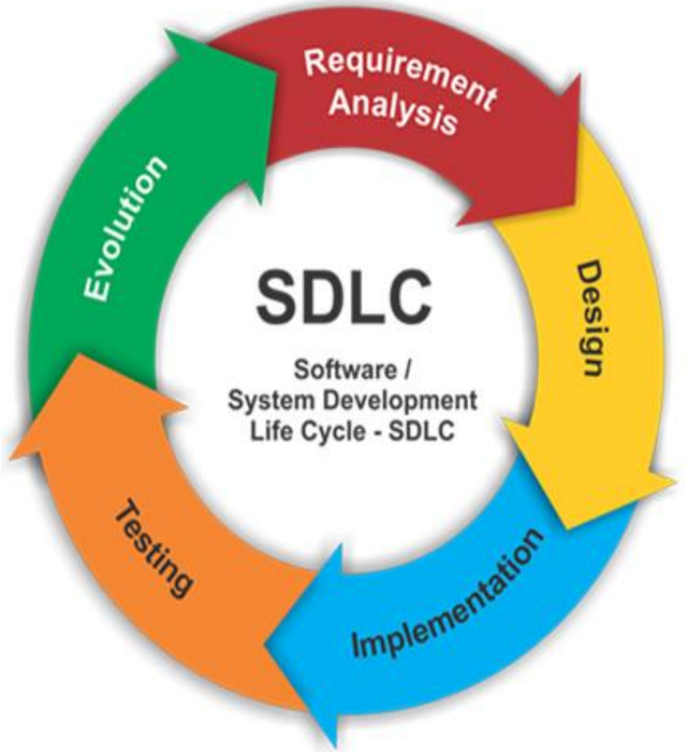
Courtesy:

Roger Pressman, Ian Sommerville &

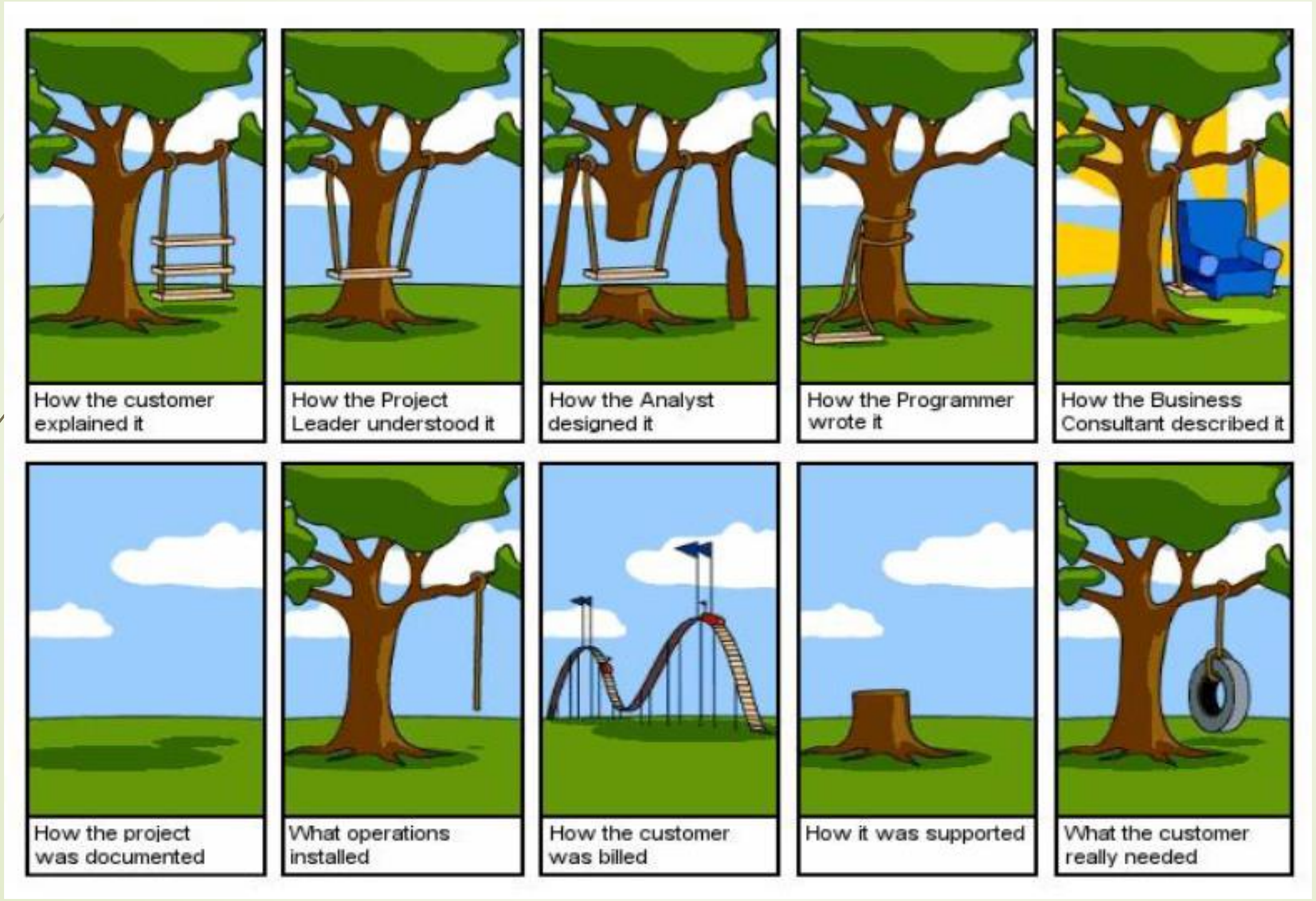
Prof Rajib Mall

Prescriptive Process Model:

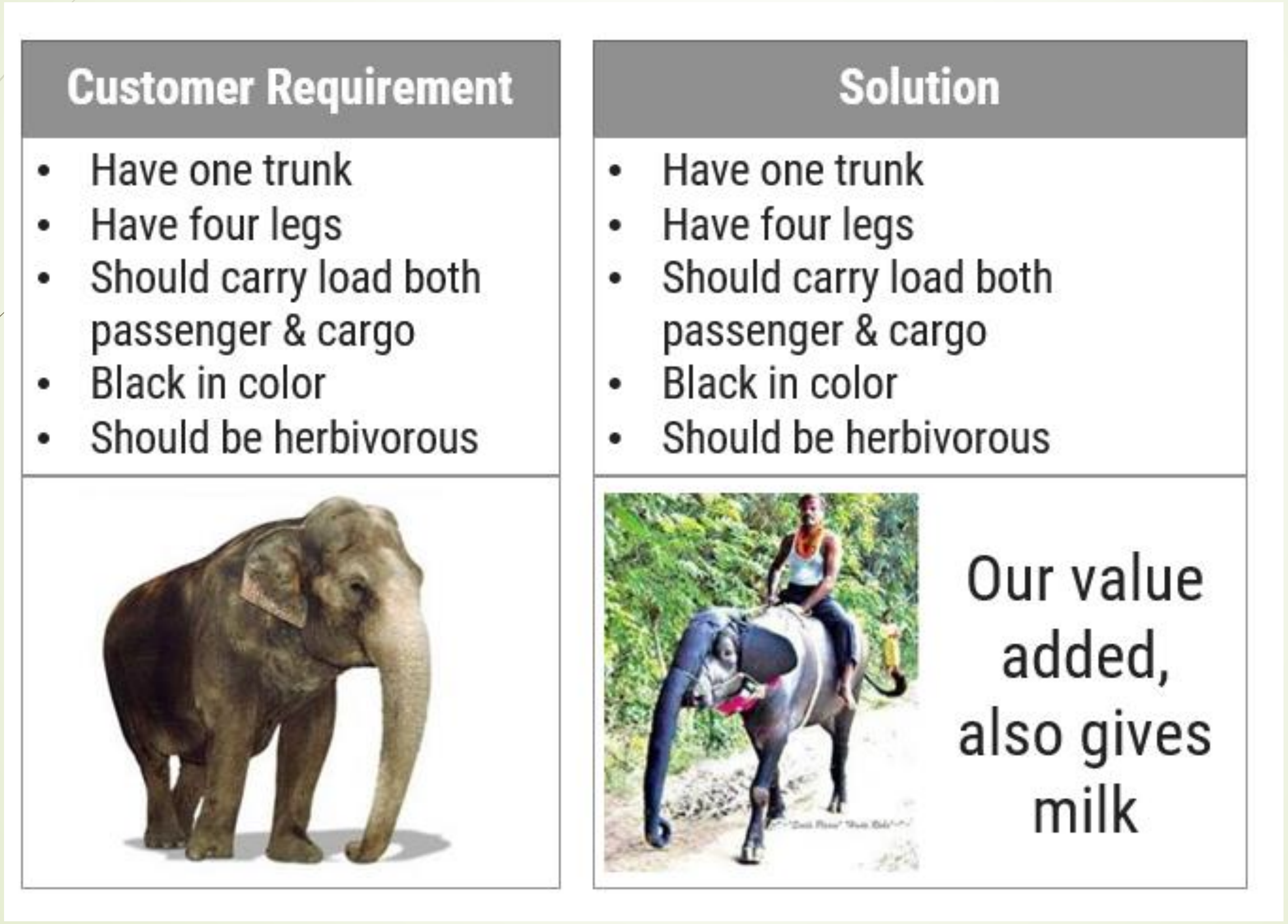
- Also known as **Software development life cycle (SDLC)** or Application development life cycle Models
- Process models **prescribe** a distinct set of **activities, actions, tasks and milestones (deliverables)** required to engineer high quality software.
- Process **models are not perfect**, but **provide roadmap** for software engineering work.
- Software models provide stability, control and organization to a process that if not managed can easily get out of control.
- Software process models are adapted (adjusted) to meet the needs of software engineers and managers for a specific project.



Software Development without SDLC

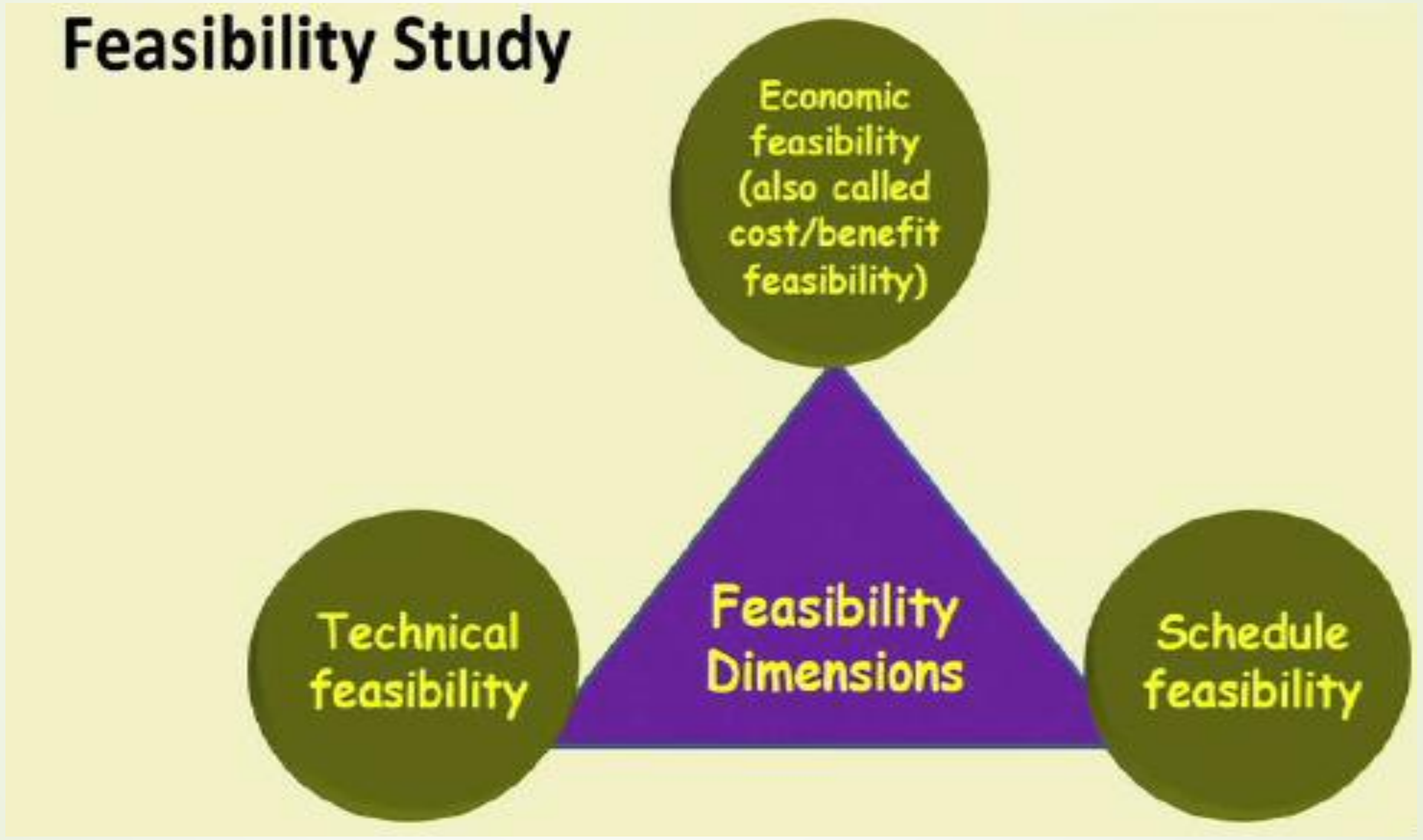

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

SDLC without Software Engineering



Customer Requirement	Solution
• Have one trunk • Have four legs • Should carry load both passenger & cargo • Black in color • Should be herbivorous	• Have one trunk • Have four legs • Should carry load both passenger & cargo • Black in color • Should be herbivorous
	Our value added, also gives milk

Feasibility Study:



Importance of Feasibility Study:

- To determine
  - Financially worthwhile
  - Technically feasible
- To understand
  - Data which would be input to the system
  - Processing needed on these data
  - Output data to be produced by system
  - Various constraints on the behavior of the system

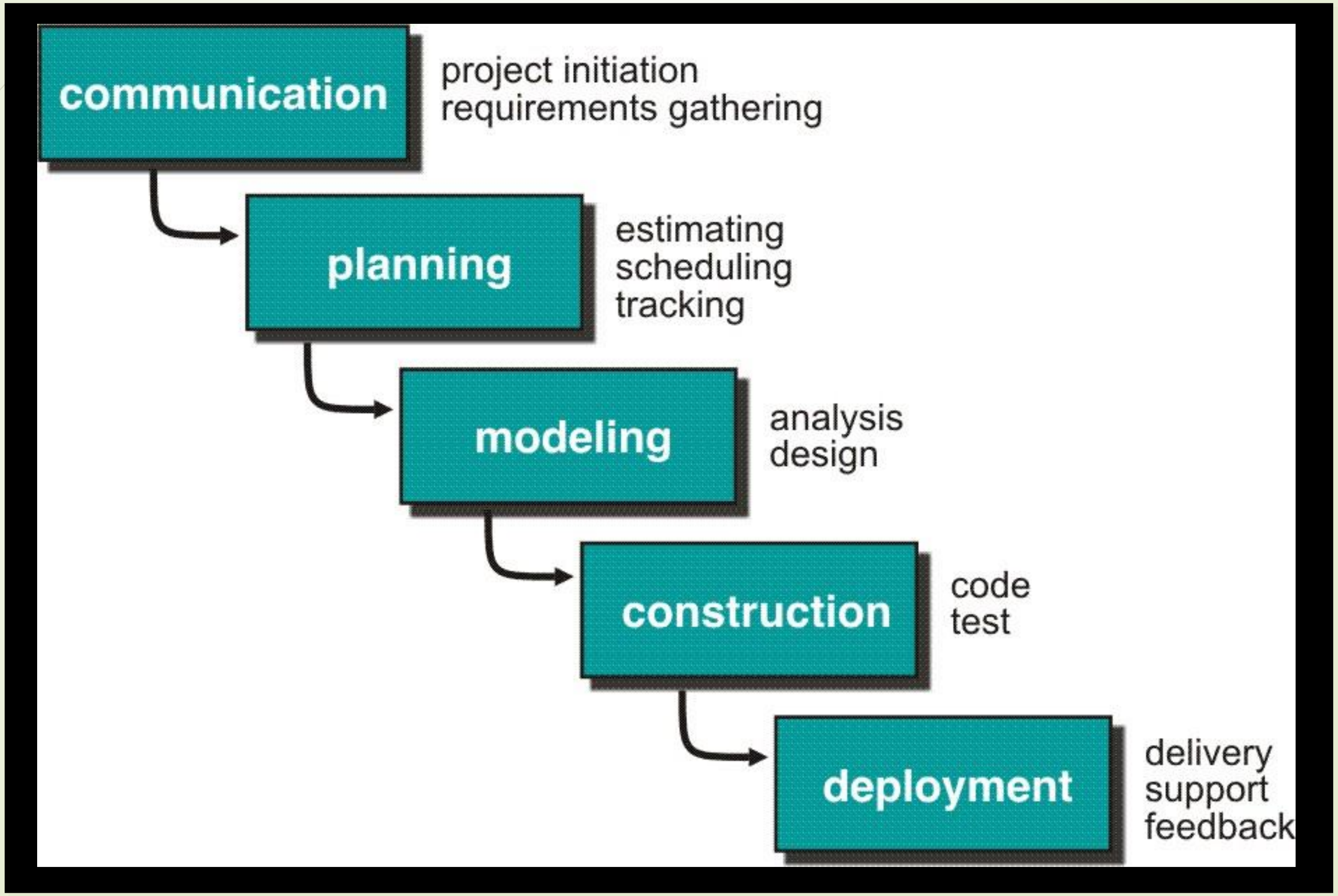
Cost Benefit Analysis:

- Identify all the costs
  - Development cost
  - Set-up cost
  - Operational cost
- Identify the value of benefits
- Check benefits are greater than costs

Different Process Models:

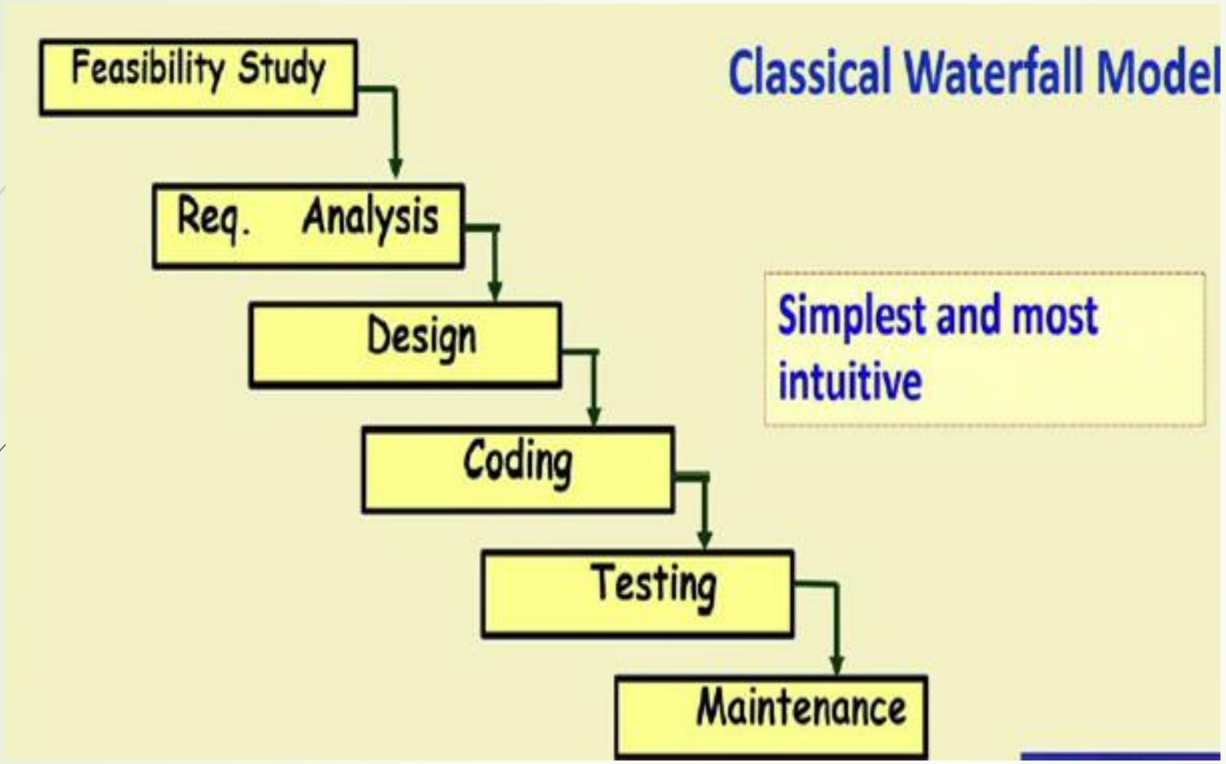
- Waterfall Model (Linear Sequential Model)
- Prototyping Model
- Incremental Process Model
- The Spiral Model
- Rapid Application Development Model
- Agile Model

The Waterfall Model:



The Waterfall Model

Classic life cycle or linear sequential model



When requirements for a problems are well understood then this model is used in which work flow from communication to deployment is linear.

The Waterfall Model: Deliverables

- Project plan and feasibility report
- Requirements documents
- System design documents
- Test plans and test reports
- Source code
- Software manuals (User manual, installation manual)

The Waterfall Model: Advantages

- Easy to understand, easy to use
- Provides a reference to inexperienced staff
- Provides requirement stability
- Simple to implement and manage

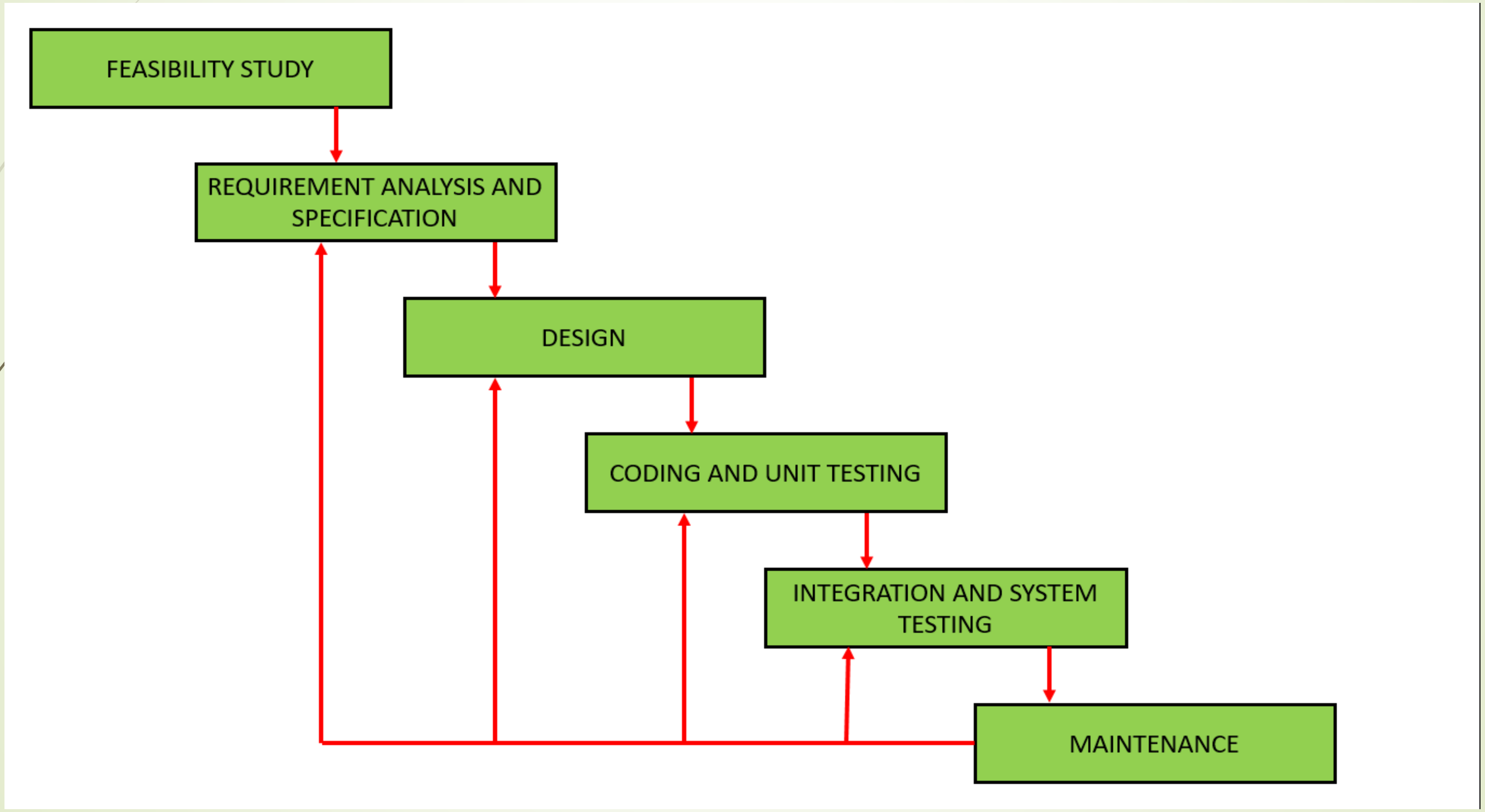
The Waterfall Model: Disadvantages

- Unable to accommodate changes at later stages, that is required in most of the cases.
- Working version is not available during development which can lead the development with major mistakes.
- Deadlock can occur due to delay in any step.
- Rigid phase sequence is not suitable for large projects.
- Cannot handle different types of risks.

The Waterfall Model: When to use?

- **Requirements** are very well **known**, **clear** and **fixed**
- Product **definition** is **stable**
- **Technology** is **understood**
- There are **no ambiguous** (unclear) **requirements**
- Ample (**sufficient**) **resources** with required **expertise** are **available** freely
- The **project** is **short**

Iterative Waterfall Model:


FEASIBILITY STUDY
REQUIREMENT ANALYSIS AND SPECIFICATION
DESIGN
CODING AND UNIT TESTING
INTEGRATION AND SYSTEM TESTING
MAINTENANCE

Phase Containment of Errors:

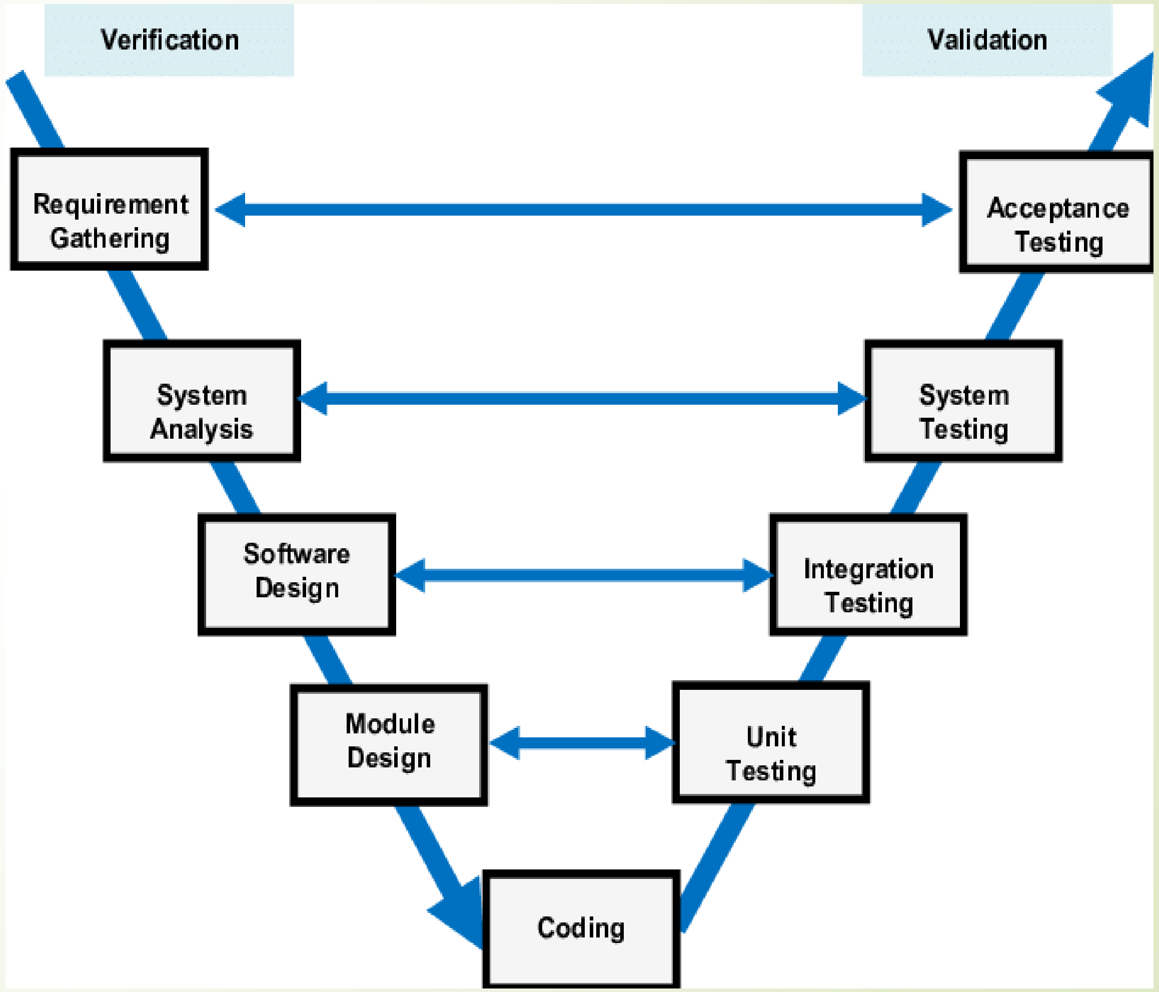
- Errors should be detected in the same phase in which they are introduced.
- The principal of detecting errors as close to its point of introduction as possible is known as Phase Containment of Errors.

Iterative Waterfall Model:

- **Advantages:**
  - Feedback Path
  - Simple and Cost Effective
  - Well Organized
- **Disadvantages:**
  - Difficult to incorporate change requests
  - Incremental delivery not supported
  - Overlapping of phases not supported
  - Risk handling not supported
  - Limited Customer Interactions

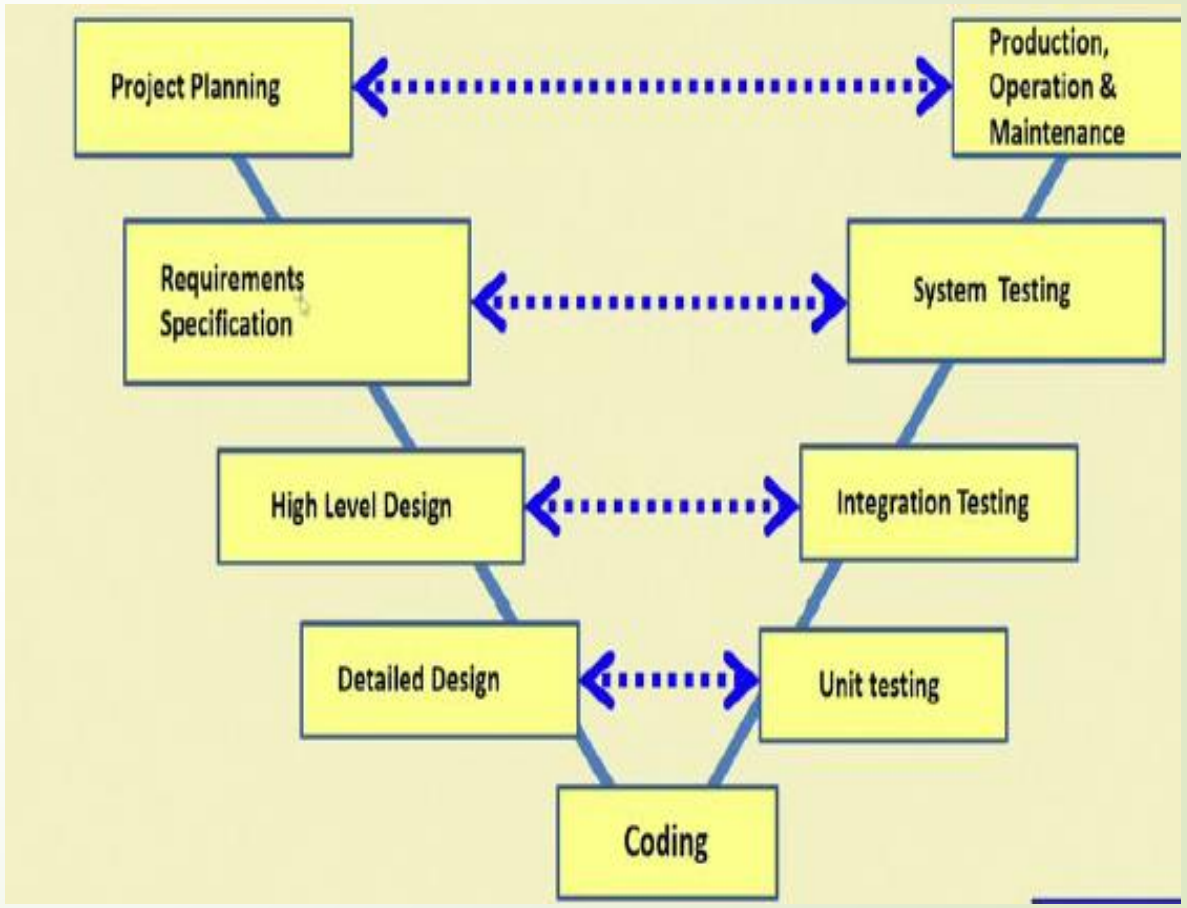
V-Model:

- Variant of Waterfall Model
  - Emphasizes on verification and validation
  - V & V activities are spread over entire life cycle.
  - V-model may be used for development of safety-critical software that are required to have high reliability.



V-Model:

- Variant of Waterfall Model
  - Emphasizes on verification and validation
  - V & V activities are spread over entire life cycle.
  - V-model may be used for development of safety-critical software that are required to have high reliability.



V-Model:

- **Advantages:**
  - Emphasizes planning for verification and validation of software
  - Each deliverable is made Testable
  - Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
  - Avoids the downward flow of the defects.
- **Disadvantages:**
  - Does not support overlapping of phases
  - Does not handle iterations and phases
  - Does not provide support for effective risk handling
  - Does not easily accommodate latter changes into requirements

Prototyping Model:

- Considered as an extension of the waterfall model.
- Before starting actual development,
  - A working prototype of the system should first be built.
- A Prototype is a toy and crude implementation of a system.

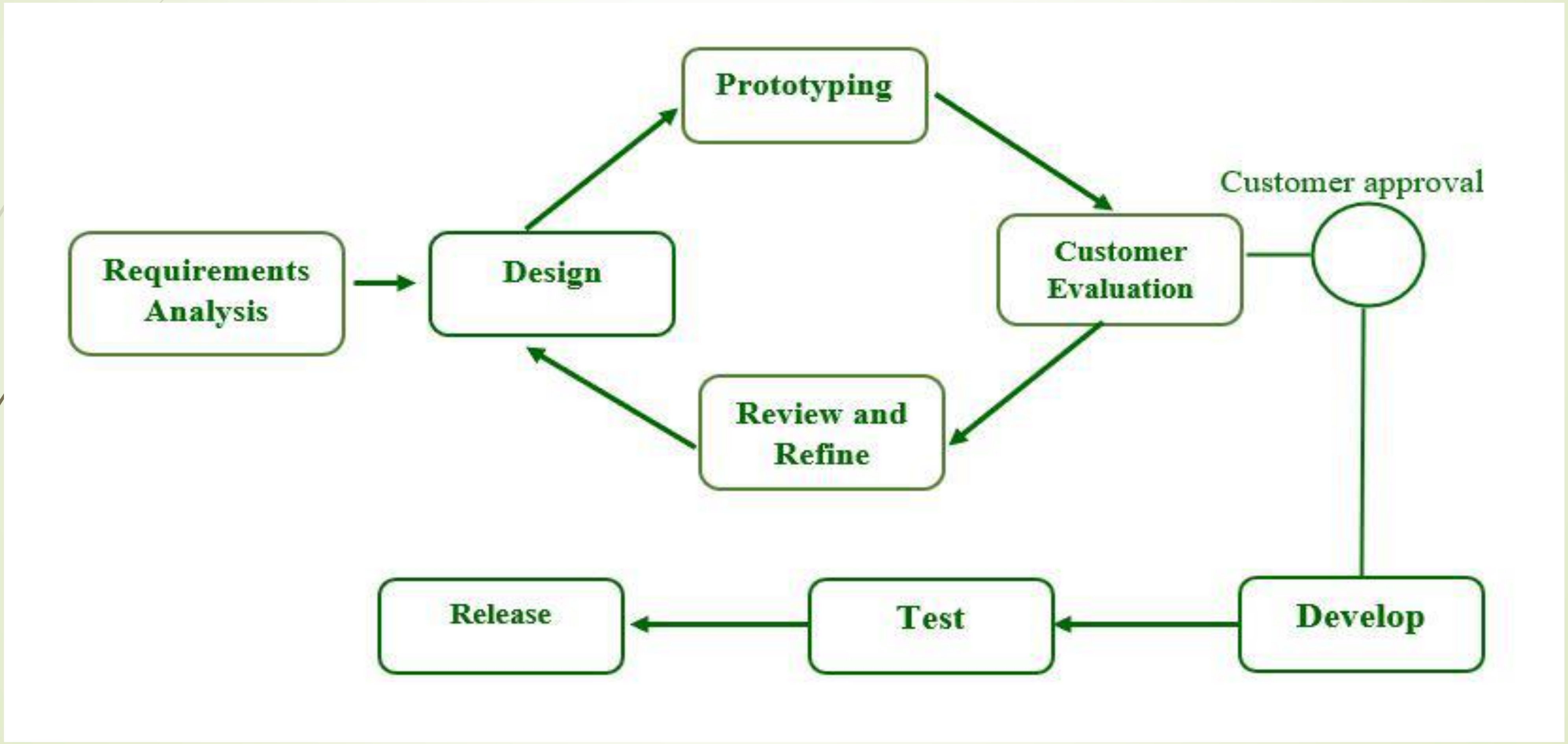
Reason for Developing Prototype:

- Learning by doing[Useful where requirements are partially known]
- Improved Communication
- Improved user involvement
- Reduced Maintenance cost
- Reduced need for documentation
- Examine Technical issues associated with product development
  - Response Time of a Hardware Controller
  - Efficiency of a sorting algorithm
- Illustrate to Customer
  - Input data formats, messages, reports, etc.

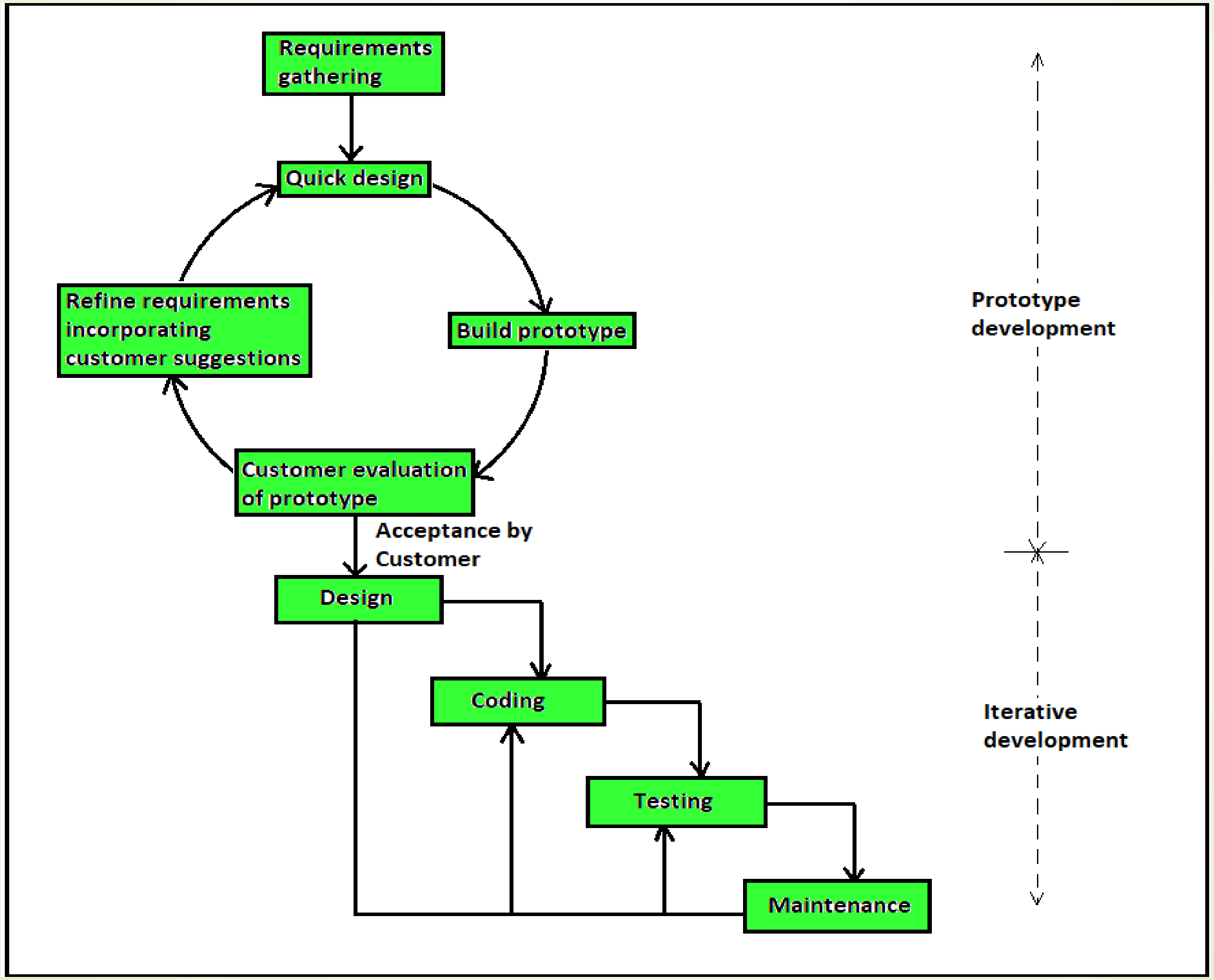
Prototyping Model:

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

Prototyping Model:



Prototyping Model:



Prototyping Model:

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

The Prototyping Model:

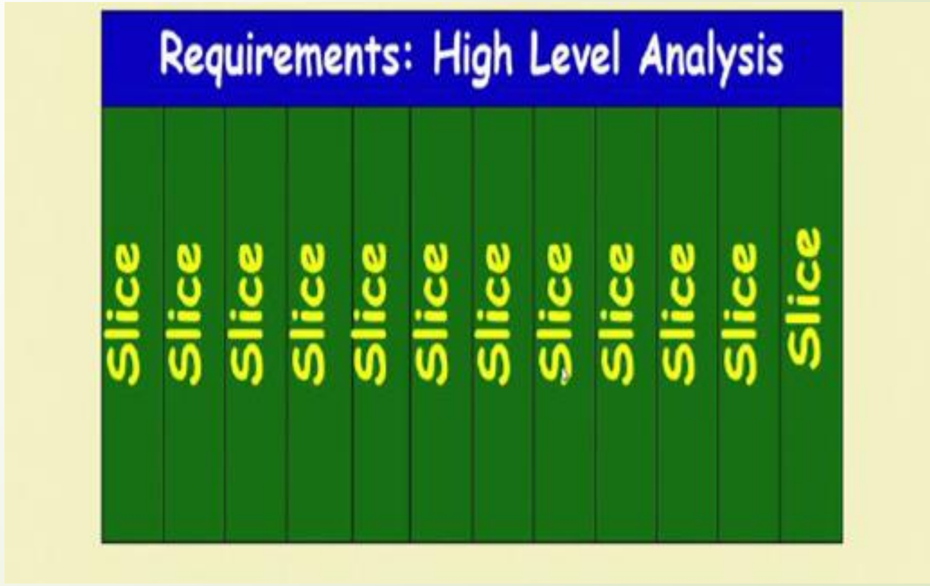
- **Advantages:**
  - The resulting software is usually more usable
  - User needs are better accommodated
  - The design is of higher quality
  - The resulting software is easier to maintain
  - Overall, the development incurs less cost
- **Disadvantages:**
  - For some projects, it is expensive
    - Susceptible to over-engineering..
    - The prototyping model is ineffective for risks identified later during the development cycle.

The Prototyping Model:

- **When to use Prototyping Model:**
  - Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
  - The GUI part of a software system is almost always developed using the prototyping model.
  - The prototyping model is useful when the exact technical solution are unclear to the development team.
  - Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model.
  - Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system.

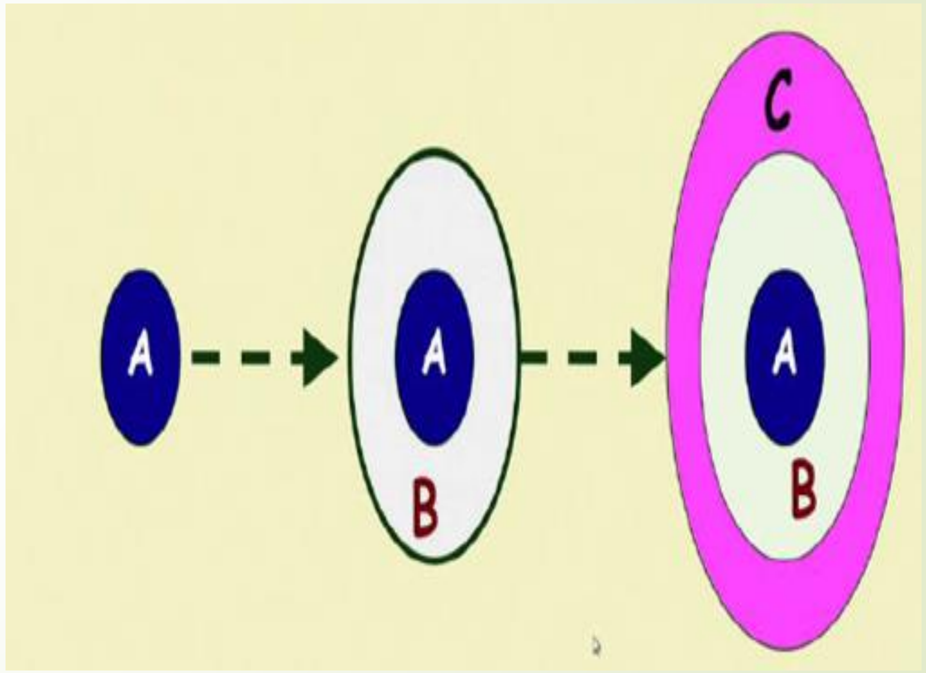
The Incremental Model:

- In incremental model the whole requirement is divided into various builds.
- Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle. Cycles are divided up into smaller, more easily managed modules.
- **Key characteristics**
  - Builds system incrementally
  - Consists of a planned number of iterations
  - Each iteration produces a working program
- **Benefits**
  - Facilitates and manages changes
  - Foundation of agile techniques



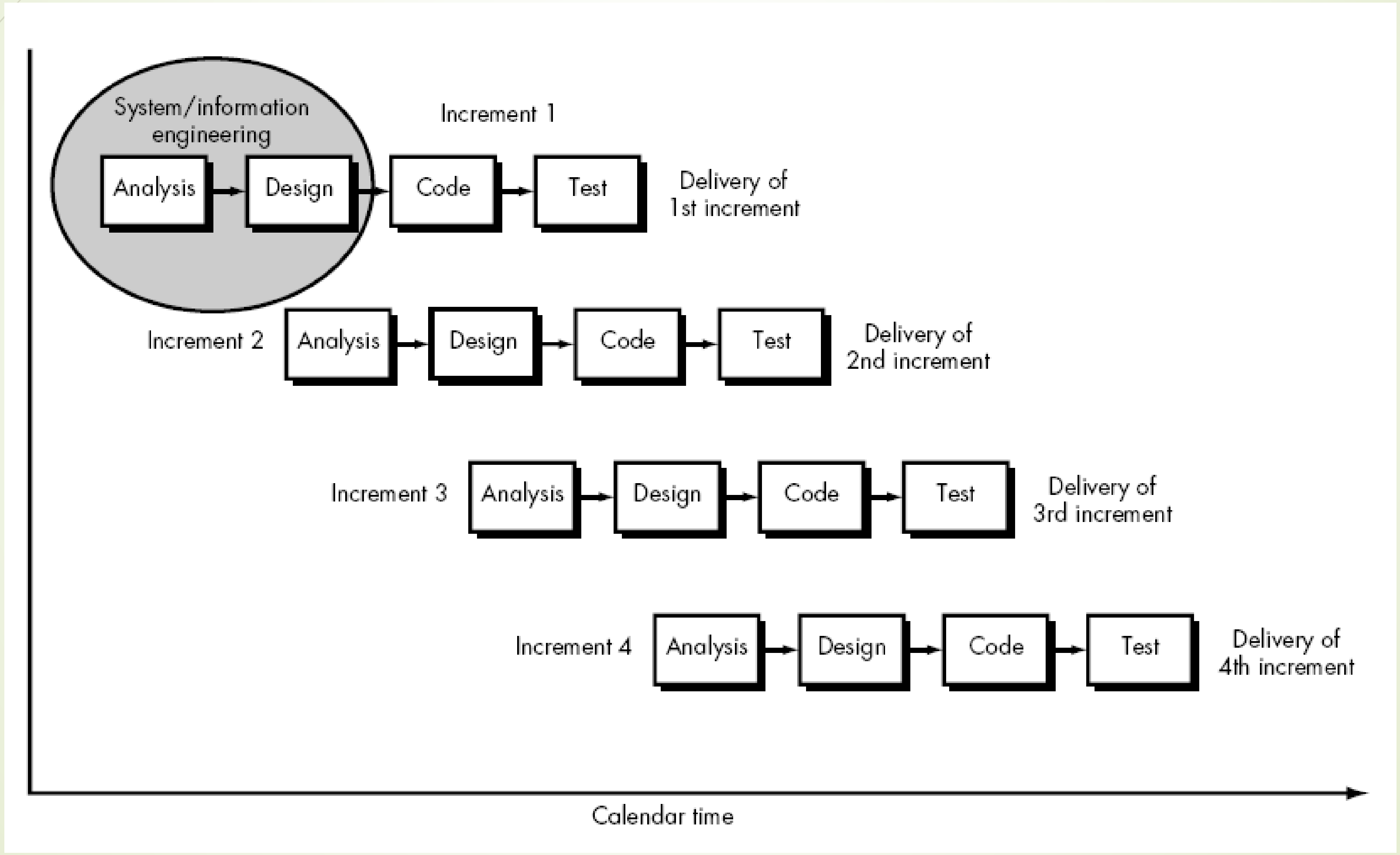
The Incremental Model:

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



A, B, and C are modules of a software product

The Incremental Model:



The Incremental Model:

- **Advantages:**
  - Error Reduction.
  - Incremental resource deployment
  - Generates working software quickly
  - Flexible to customer
  - Easier to test and debug
  - Easier to manage risk
- **Disadvantages:**
  - Needs good planning and design.
  - Needs a clear and complete definition of the complete system before it can be broken down and built incrementally.
  - Integration needs are very high
  - Total cost is higher than waterfall.

Evolutionary Model with Iteration:

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

Evolutionary Model with Iteration:

- "A complex system will be most successful if implemented in small steps… "retreat" to a previous successful step on failure… opportunity to receive some feedback from the real world before throwing in all resources… and you can correct possible errors…" **-Tom Glib in Software Metrics**

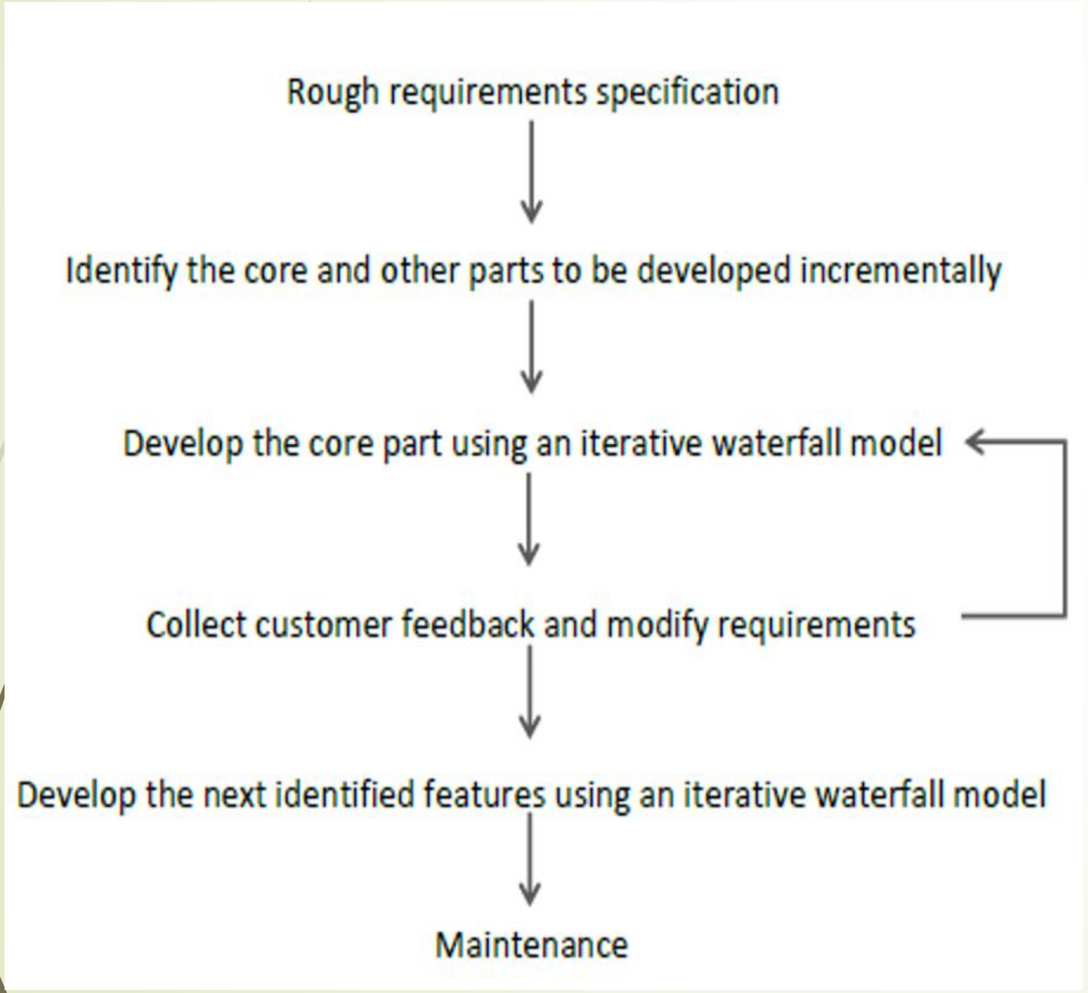
Evolutionary Model with Iteration:

- First develop the core modules of the software.
- The initial skeletal software is refined into increasing levels of capability: (Iterations)
  - By adding new functionalities in successive versions.
- **Activities in an Iteration**
  - Software developed over several "mini waterfalls".
- The result of a single iteration:
  - Ends with delivery of some tangible code
  - An incremental improvement to the software ---leads to evolutionary development

Evolutionary Model with Iteration:

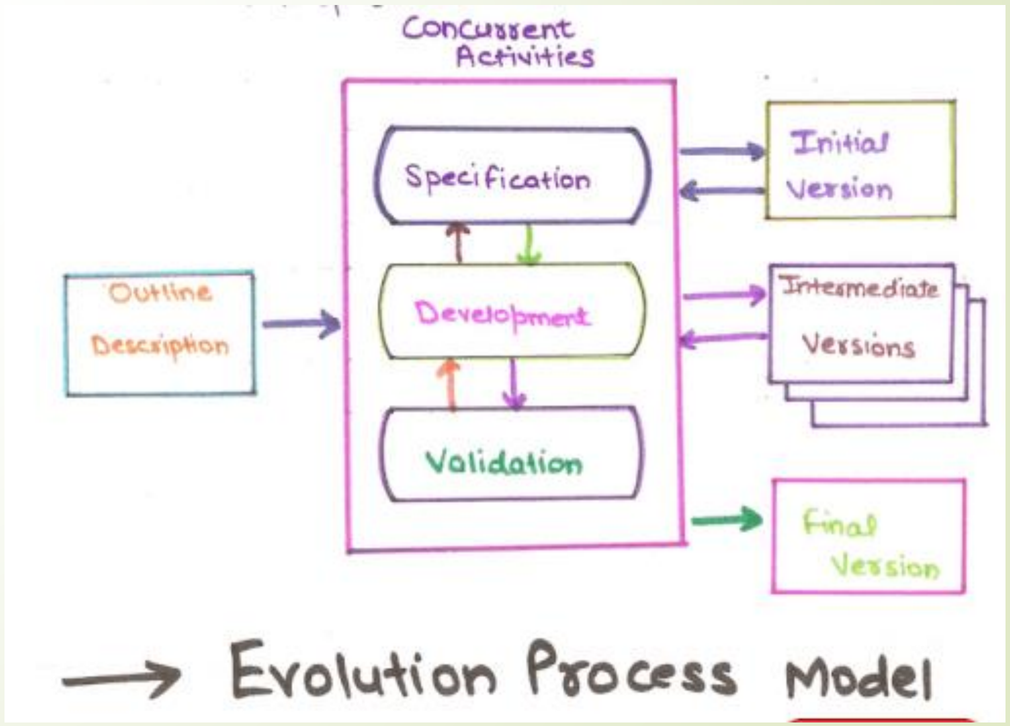
- Outcome of each iteration: tested, integrated, executable system
- Iteration length is short and fixed
  - Usually between 2 and 6 weeks
  - Development takes many iterations (for example: 10-15)
- Does not "freeze" requirements and then conservatively design :
  - Opportunity exists to modify requirements as well as the design…
- Successive versions:
  - Functioning systems capable of performing some useful work.
  - A new release may include new functionality:
  - Also existing functionality in the current release might have been **enhanced.**

Evolutionary Model with Iteration:


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
Develop the next identified features using an iterative waterfall model
Maintenance

Evolves an initial implementation with user feedback:

–Multiple versions until the final version.



Evolutionary Model: Advantages

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

Evolutionary Model: Disadvantages

- Feature division into incremental parts can be non-trivial.
  - **Functional dependency, small sized projects.**
- The process is intangible:
  - **No regular, well-defined deliverables.**
- The process is unpredictable:
  - **Hard to manage, e.g., scheduling, workforce allocation, etc.**
- Systems are rather poorly structured:
  - **Continual, unpredictable changes tend to degrade the software structure.**

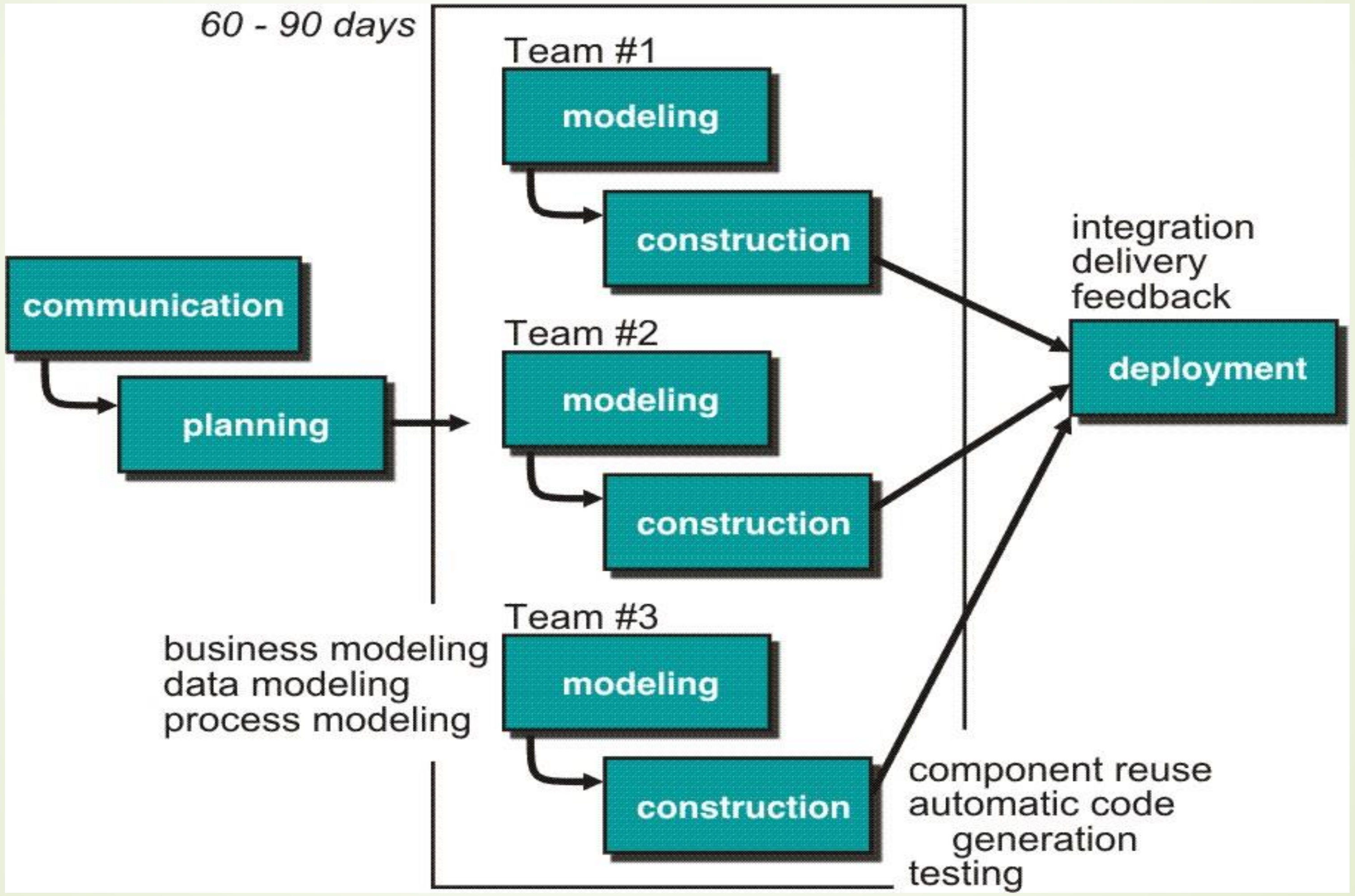
RAD Model:

- Rapid Application Development (RAD) Model
- Sometimes referred to as the **rapid prototyping model.**
- **Major aims:**
  - Decrease the time taken and the cost incurred to develop software systems.
  - Facilitate accommodating change requests as early as possible:
    - Before large investments have been made in development and testing.
- A way to reduce development time and cost, and yet have flexibility to incorporate changes:
  - Make only short term plans and **make heavy reuse of existing code.**
  - Plans are made for one increment at a time.
- The time planned for each iteration is called a time box.
- Each iteration (increment):
  - Enhances the implemented functionality of the application a little.

RAD Model:

- During each iteration,
  - A quick-and-dirty prototype-style software for some selected functionality is developed.
  - The customer evaluates the prototype and gives his feedback.
- The prototype is refined based on the customer feedback.
- How Does RAD Facilitate Faster Development?
  - RAD achieves fast creation of working prototypes.
  - Through use of specialized tools.
- These specialized tools usually support the following features:
  - Visual style of development.
  - Use of reusable components.
  - Use of standard APIs

RAD Model:



60 - 90 days
Team #1
modeling
construction
Team #2
modeling
construction
Team #3
modeling
construction
communication
planning
business modeling
data modeling
process modeling
integration
delivery
feedback
deployment
component reuse
automatic code generation
testing

RAD Model:

- **Advantages of the RAD model:**
  - Reduced development time
  - Increases reusability of components
  - Quick initial reviews occur
  - Encourages customer feedback
- **Disadvantages of RAD model:**
  - Only system that can be modularized can be built using RAD
  - Requires highly skilled developers/designers.
  - High dependency on modelling skills
  - Inapplicable to cheaper projects as cost of modelling and automated code generation is very high.

RAD Model:

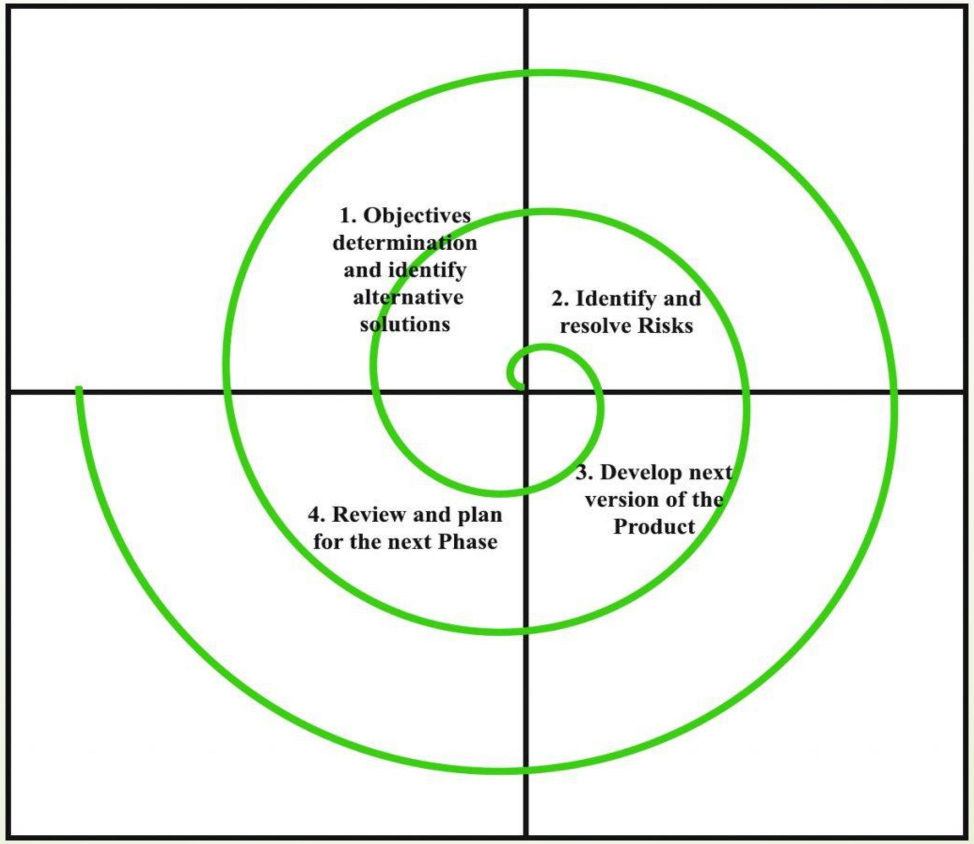
- **For which Applications is RAD Suitable?**
  - Customized product developed for one or two customers only
  - Performance and reliability are not critical.
  - The system can be split into several independent modules.
  - When a system needs to be produced in a short span of time (2-3 months)
  - When a budget is high enough to afford designers for modelling along with the cost of automated tools for code generation
- **For Which Applications RAD is Unsuitable?**
  - High performance or reliability required
  - No precedence for similar products exists
  - The system cannot be modularized.

Spiral Model:

- Proposed by Boehm in 1988.
- **Risk driven** software development model
  - Technology is new
  - Research work is required
  - Lack of crystal clear requirements
  - Critical timeline
- Features of **Incremental** model and **Evolutionary** model
- Requirement **Prioritization**
- **Good feedback mechanism** based approach.
- Required **good planning.**

Spiral Model:

- Risk Handling with prototype
- The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

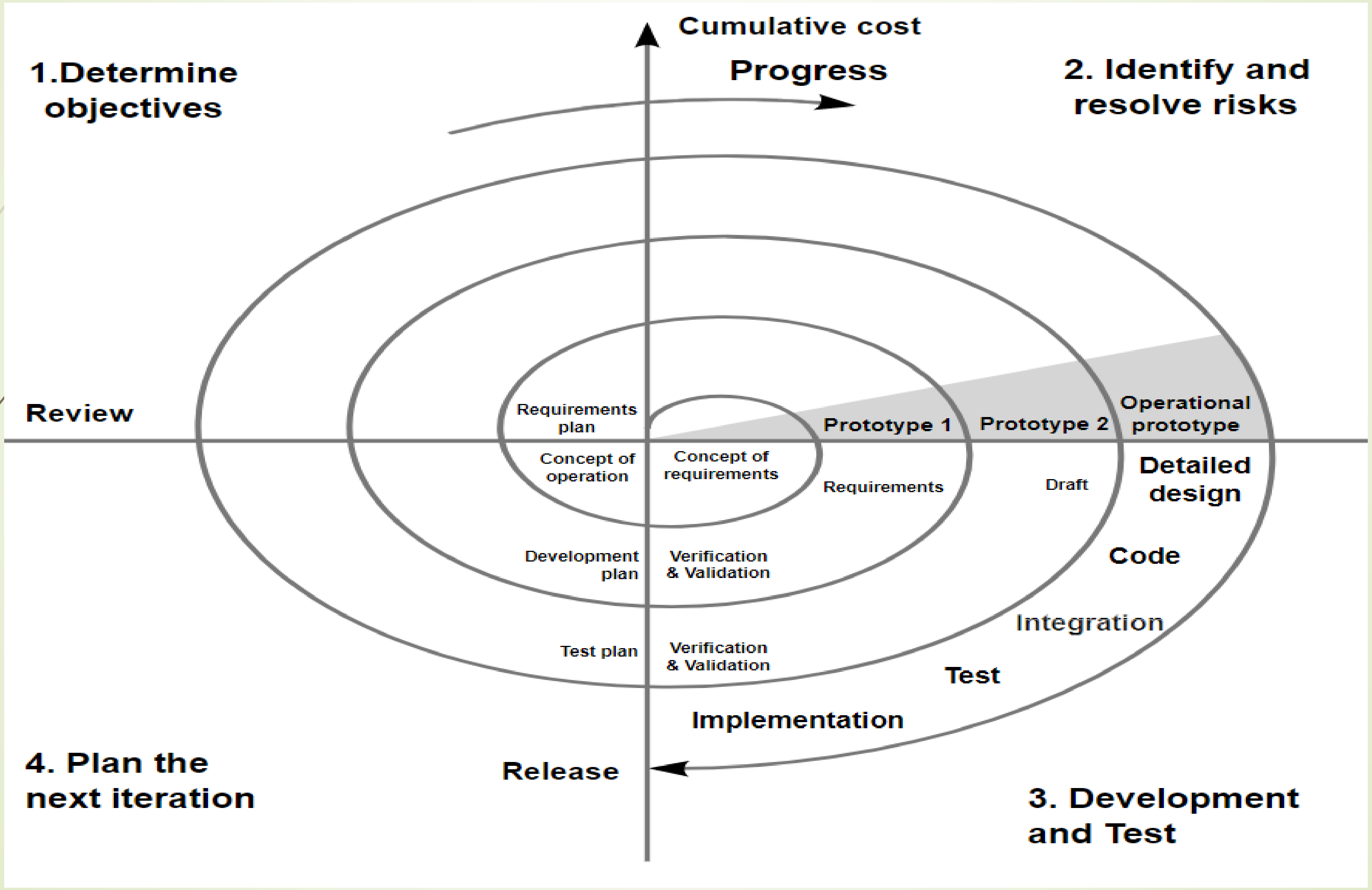
Spiral Model:

- Each loop in the spiral is split into four sectors (quadrants).
- **Objectives determination and identify alternative solutions:**
  - Defining objective of specific release and what will be its success criteria?
  - Requirement Gathering
  - Functionality
  - Performance
  - Identifying the alternatives and the constraints.
- **Identify and resolve Risks**
  - Detailed analysis of the identified feature is carried out. and After that the risk is identified through building a prototype.
  - Reviewing and validating objectives, alternatives and constraints.
  - Identifying and Resolving Risk: New Technology, Lack of Experience, Tight Schedule, Cost overrun, Resource constraints.

Spiral Model:

- **Develop next version of the Product**
  - after the risk is resolved the development occurs
  - Activities carried out are: Detailed Design, Design Review, Coding, Code Review, Testing and Integration.
- **Review and plan for the next Phase**
  - review the previous phase for its completion and planning of next phase occurs.
  - Collecting backlogs and reviewing them
  - Develop and review project plan
  - Provide support and software installation if required.
- Every spiral or every phase may not lead to a deployable software at the client site.

Spiral Model:



Spiral Model:

- The exact number of loops of the spiral is unknown and can vary from project to project.
- 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.

Risk Handling in Spiral Model:

- A risk is any adverse situation that might affect the successful completion of a software project.
- The most important feature of the spiral model is handling these unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype. The spiral model supports coping up with risks by providing the scope to build a prototype at every phase of the software development.
- The Prototyping Model also supports risk handling, but the risks must be identified completely before the start of the development work of the project.
- But in real life project risk may occur after the development work starts, in that case, we cannot use the Prototyping Model. In each phase of the Spiral Model, the features of the product dated and analyzed, and the risks at that point in time are identified and are resolved through prototyping. Thus, this model is much more flexible compared to other SDLC models.

Spiral Model:

- **Advantages of Spiral Model:**
  - Risk Handling
  - Good for large projects
  - Flexibility in Requirements
  - Customer Satisfaction
  - Meta Model
- **Disadvantages of Spiral Model:**
  - Complex
  - Expensive
  - Too much dependability on Risk Analysis
  - Difficulty in time management

Prescriptive Process Models:

- **Waterfall Model:**
  - Basic model and all other models as embellishments of this model.
  - Model supports no mechanism to correct the errors that are committed during any phases but detected later phase.
- **Iterative Waterfall Model:**
  - Probably the most widely used model.
  - Suitable only for well understood problems
  - Not suitable for large project and project that suffers from large number of risks.

Prescriptive Process Models:

- **Prototyping Model:**
  - Suitable for projects for which either user requirements or technical aspects are not clear.
  - Popular to develop user interface part of project.
  - All the risks are tried to identify before the project starts. However, fails to manage risks during the development.
- **Incremental Model:**
  - Incremental Process models are iterative in nature and produce working versions of the software quite rapidly.
  - The complete system understanding is required with increments to be developed.

Prescriptive Process Models:

- **Evolutionary Model:**
  - Suitable for large problems which can be decomposed into set of modules for incremental development and delivery.
  - Only used when incremental delivery of the system is acceptable to the customer.
- **RAD Model:**
  - When a system needs to be produced in a short span of time.
  - Make only short term plans and make heavy reuse of existing code.
  - Does not support to the project with High budget, No precedence for similar products exists and the system cannot be modularized.

Prescriptive Process Models:

- **Spiral Model:**
  - The spiral model is considered a meta model and encompasses all other model.
  - Suitable for development of technically challenging and large software that are prone to several kinds of risk.
  - Complex and expensive model.

Agile Model
and
Agile vs Structured Process Models

Why do we need Agile ?

- The frequent change in the requirement during the development escalates cost in traditional process model.
- Rapid shift from development of software product to the development of customized software and the increased emphasis and scope for reuse.
- It is time consuming to push the changes in the monolithic applications (Schedule downtime).
- Traditional model focus more on documentation and usage of tools.
- Waterfall model prescribes almost no customer interactions after the requirement analysis.

Evolution of Agile Process:

- Mid 1990s – prescribed means of correct software development was
  - a heavyweight software development methodology following a **"Do it right first time"** philosophy…
  - a highly compartmentalized approach based on
    - the belief that when one **does not adhere to a methodology**, projects run into **troubles**
- Fact of the matter was the otherwise !!!
  - an exceedingly iterative, lightweight software development methodology was being followed in practice.

The Agile Process:

- 'Agile' means 'Light Weight' methods
  - evolved in the 1990s when developers decided to break away from traditional structured, segmented, bureaucratic approaches to software development
    - to move towards more flexible development styles - a number of attempts were made to suggest alternatives
  - some of the most prominent/popular agile software development methods
    - Extreme Programming – in 1999, first one to attract wide attention
    - ‘Scrum' in 1995,
    - 'Crysta' , 'Adaptive Software Development', 'Dynamic Systems Development Method' in 1995
    - 'Feature Driven Development‘….

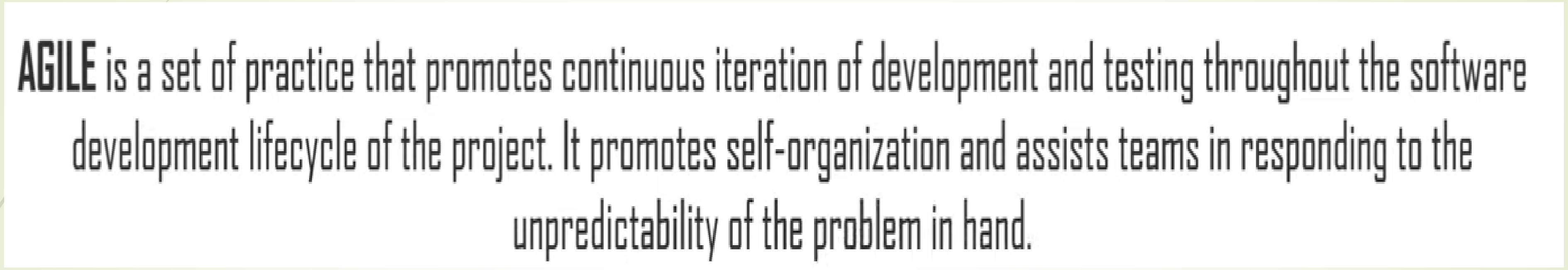
The Agile Process…

- Unlike the **'set-in-stone'** approach of waterfall development models
- The agile models
  - focuses on **'agility'** and **'adaptability'** in development.
  - involve **multiple iterative development** schedules that seek to improve the output with every iteration.
    - each iteration goes through the all the steps of design, coding and testing.
    - the **design is not set in stone**, never totally frozen and **is kept open to last minute changes** due to iterative implementation.
  - focus on **cross functional** team structure, closely knit and self-organizing.
    - but is **allowed to evolve** as new ideas come in with each release
  - less **importance is given to documentation** than **speed of delivering** a working program.

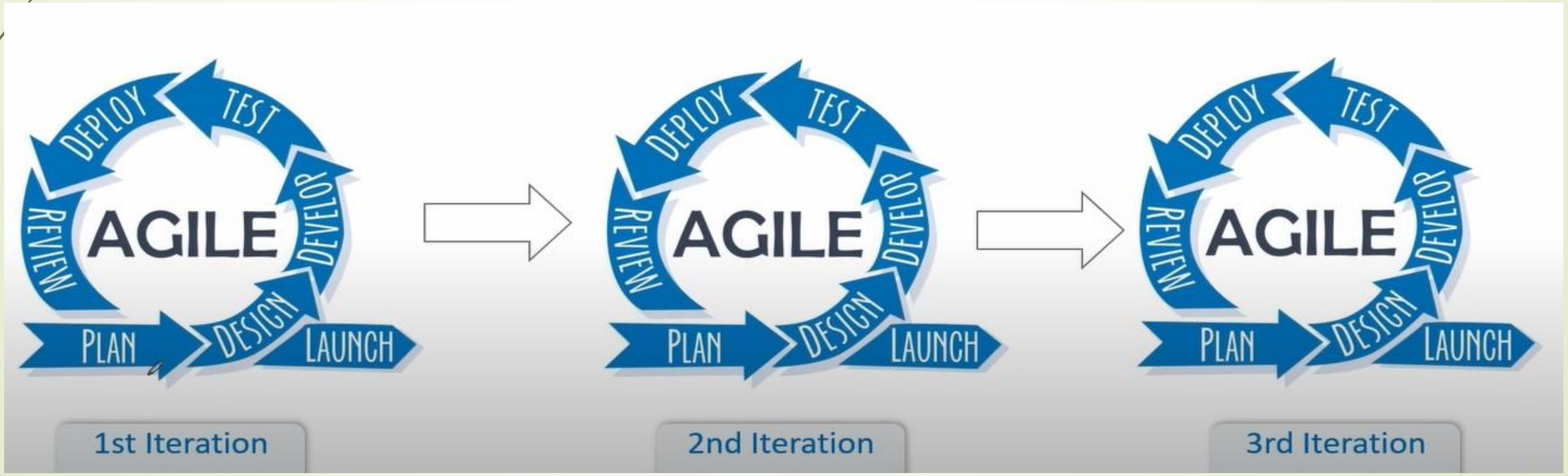
Agile Model:

- Agile: Fast Development
- Agility
  - Fitting the process to the project
  - Avoidance of things that waste time
- Designed to overcome the shortcomings of waterfall model
- The Agile model was primarily designed to help a project to adapt to change requests quickly
- In the Agile model
  - The requirements are decomposed into many small incremental parts that can be developed over one to four weeks each

Agile Model:



AGILE is a set of practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. It promotes self-organization and assists teams in responding to the unpredictability of the problem in hand.



Principles of Agile Model:

- It emphasizes on having efficient team members and enhancing communications among them is given more importance.
- At a time, only one increment is planned, developed, deployed at the customer site.
  - No long-term plans are made.
- An iteration may not add significant functionality,
  - But still a new release is invariably made at the end of each iteration
  - Delivered to the customer for regular use.
- The primary measure of progress:
  - **Incremental release of working software**
- Frequent delivery of versions ---once every few weeks.
- Requirements change requests are easily accommodated.
- Close cooperation between customers and developers.

The Agile Manifesto

- The Agile Manifesto
  - In 2001, a group of 17 pioneers in agile software development came together – many listed earlier on a slide
  - declared the 'Agile Manifesto' - a set of canonical rules of sorts for, agile software development methods.

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items not emboldened above, we value the items in bold.

The Agile Manifesto…

- Twelve principles
  - Customer satisfaction by rapid delivery of useful software
  - Welcome changing requirements, even late in development
  - Working software is delivered frequently (weeks rather than months)
  - Working software is the principal measure of progress
  - Sustainable development, able to maintain a constant pace
  - Close, daily co-operation between business people and developers
  - Face-to-face conversation is the best form of communication (co-location)
  - Projects are built around motivated individuals, who should be trusted
  - Continuous attention to technical excellence and good design
  - Simplicity
  - Self-organizing teams
  - Regular adaptation to changing circumstances

Extreme Programming Model:

- Extreme programming (XP) is one of the most important software development frameworks of Agile models.
- Kent Beck, Ward Cunningham and Ron Jeffries formulated extreme Programming in 1999.
- Based on simple Philosophy: **"If something is known to be beneficial, why not put it to constant use?"**
- **Good practices need to be practiced in extreme programming**

Extreme Programming: Activities:

- **Code Review:**
  - Helps to detect and correct problems most efficiently.
  - XP suggests pair programming as the way to achieve continuous review.
- **Testing:**
  - Helps to remove bug and improves reliability.
  - XP suggests Test Driven Development (TDD) to continually write and execute test cases.
  - Test cases are written based on user stories.

Extreme Programming: Good Practices:

- **Incremental Development:**
  - Helps to get customer feedback.
  - Extent of features delivered is a reliable indicator of progress.
  - It suggest team should come up with new increments every few days.
- **Simplicity:**
  - One should try to create a simplest code and make the basic functionality being written to work.
  - Once the simplest code works other desirable aspects can be introduced through refactoring.

Extreme Programming: Good Practices:

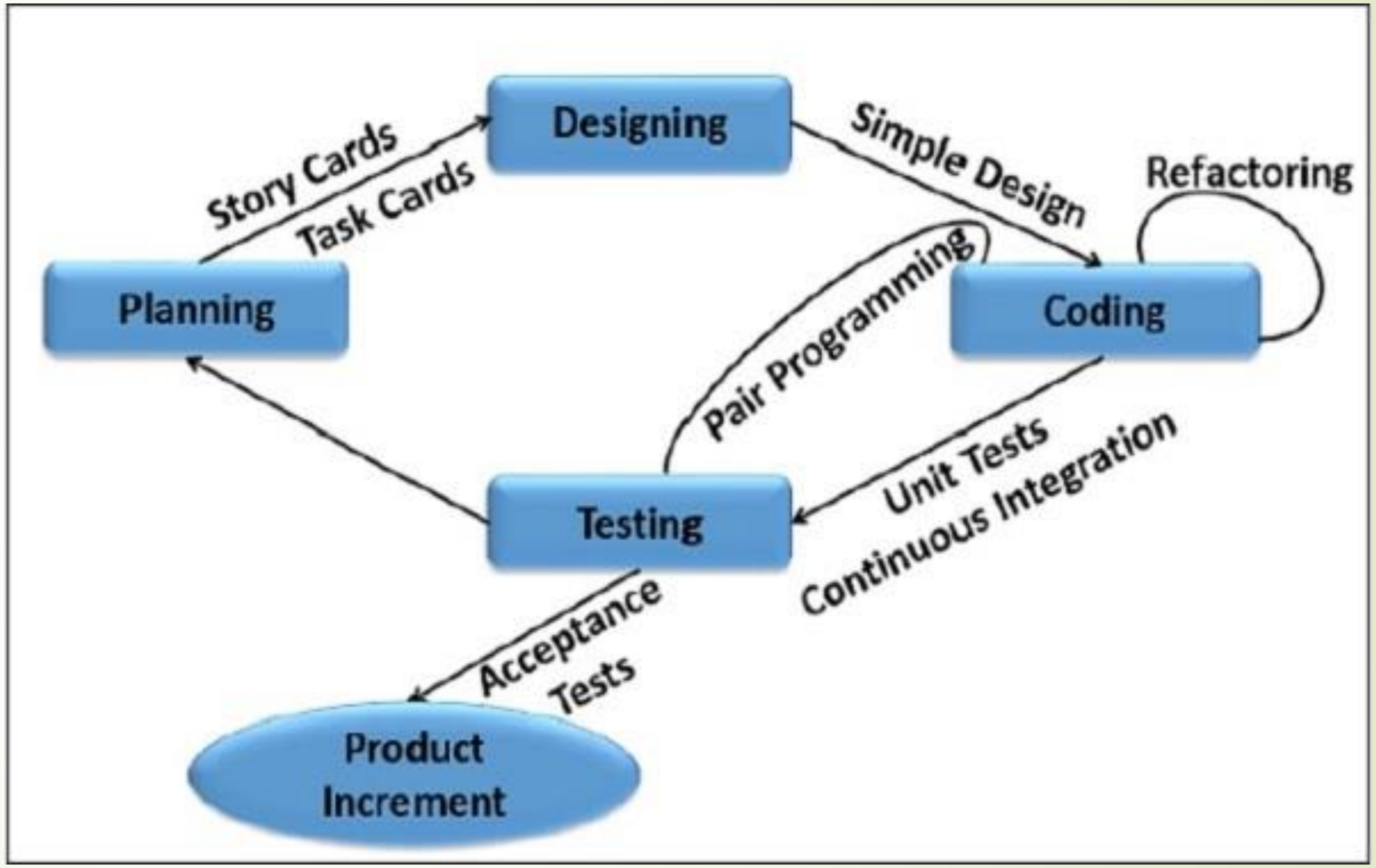
- **Design:**
  - Having good quality design is important to develop a good quality solution.
  - Team member should perform design on daily basis.
  - Achieved through refactoring, which involves improving piece of working code for efficiency and maintainability.
- **Integration Testing:**
  - Helps to identify the bugs at the interfaces of different functionalities.
  - XP suggests that the developers should achieve continuous integration.

Basic Idea of Extreme Programming Model:

- XP is based on frequent releases (called iterations), during which the developers implement "user stories".
- A user story is the conversational description by the user about feature of the required system.
  - For Example, a set of user stories about a library software can be:
    - A library member can issue a book
    - A library member can query about the availability of a book.
    - A library member should be able to return a borrowed book.
- Based on 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 can be considered to be similar to a prototype.

Extreme Programming: Activities:

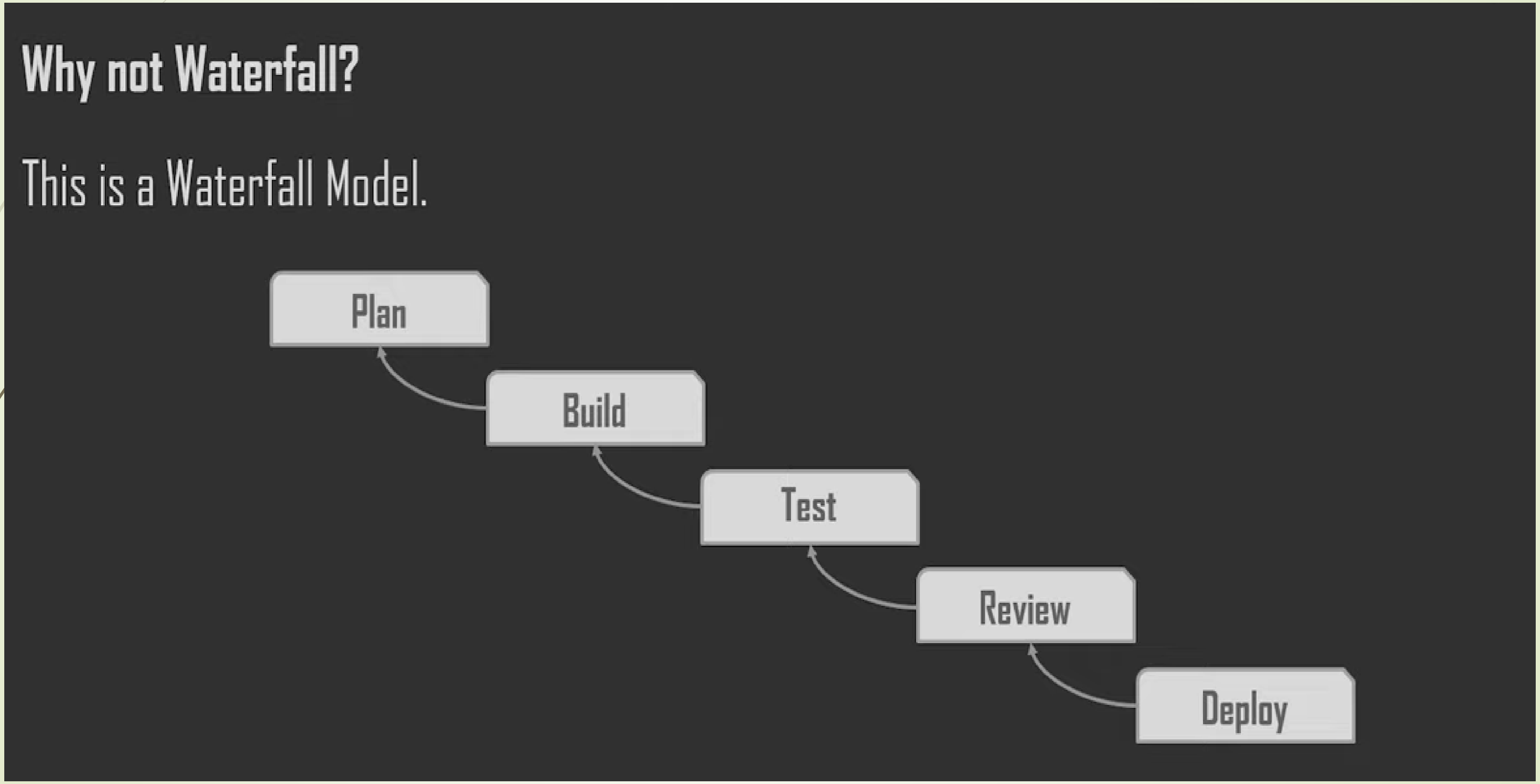
- Coding
- Testing
- Listening
- Designing
- Feedback
- Simplicity



Extreme Programming Model:

- Applicable:
  - Projects involving new technology or research projects
  - Small Projects
- Not Suited:
  - Stable requirements
  - Mission critical or safety critical systems

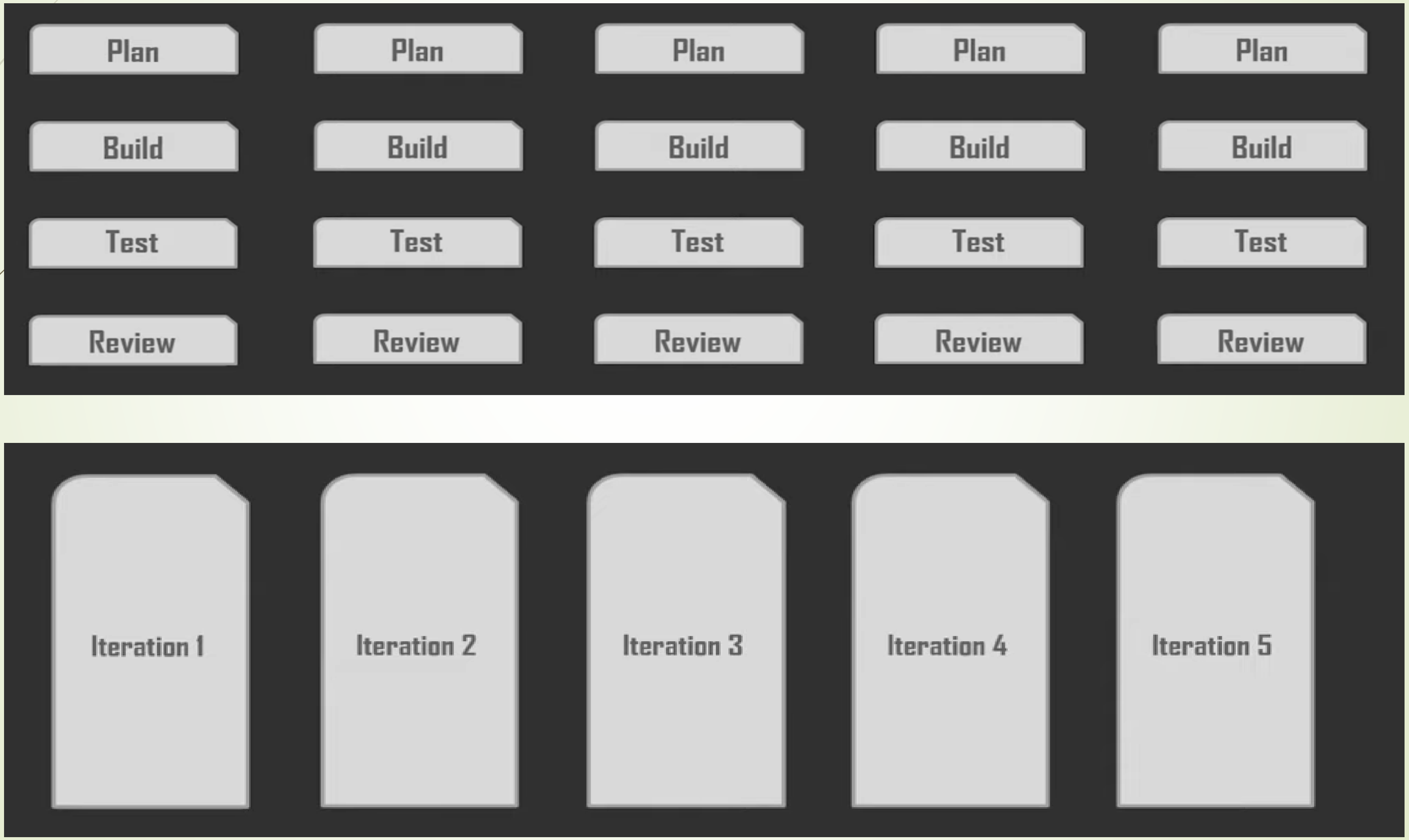
Scrum Model:



Why not Waterfall?

This is a Waterfall Model.

Scrum Model:


Plan
Build
Test
Review
Plan
Build
Test
Review
Plan
Build
Test
Review
Plan
Build
Test
Review
Plan
Build
Test
Review
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5

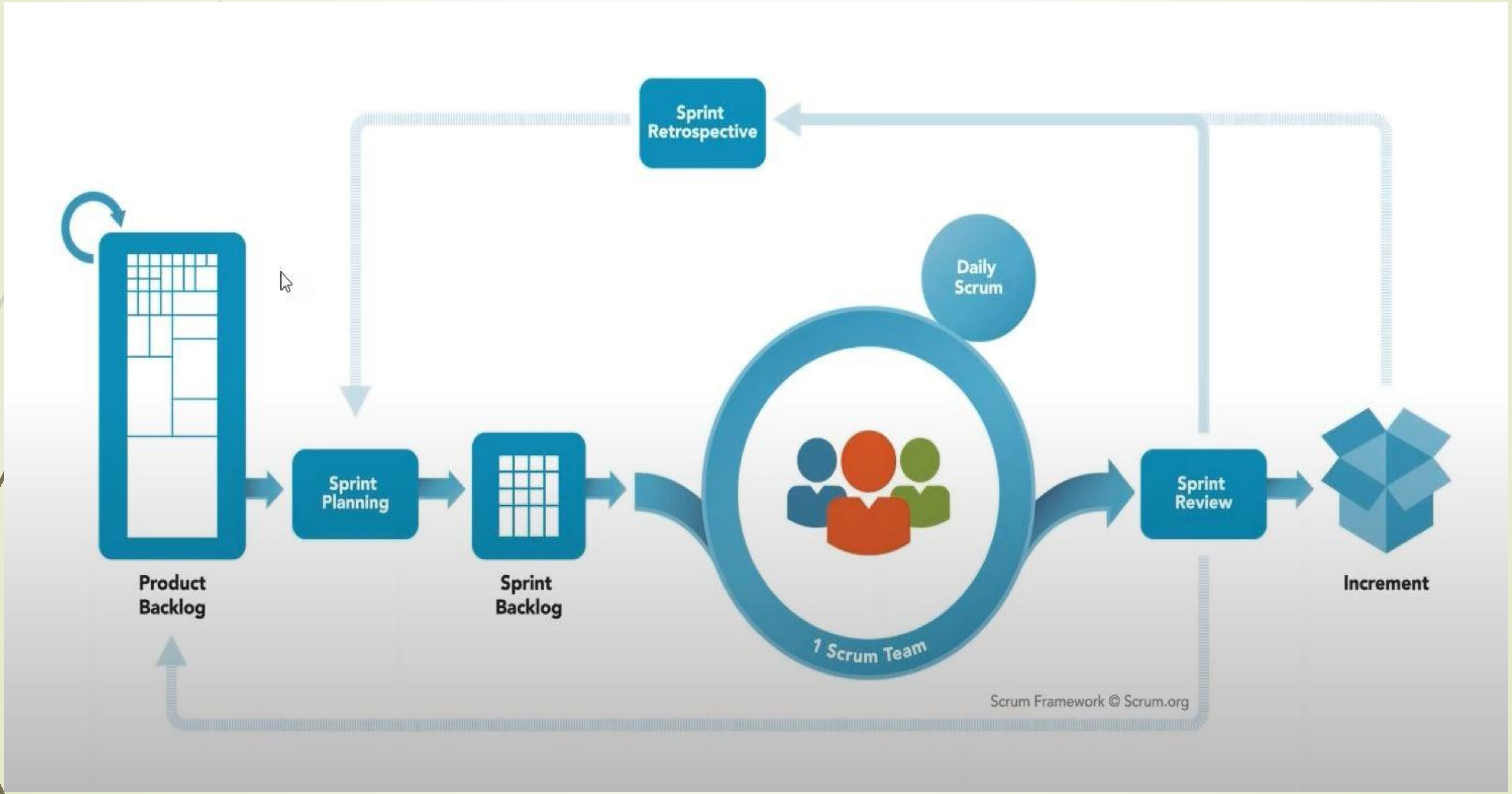
Scrum Model:

- Scrum is a framework in within which people can address complex adaptive problems while productively and creatively delivering products of the highest possible value.
  - Scrum is lightweight and is simple to understand
- Scrum promotes developing products through processes, techniques and practices with iterations and increments to deliver maximum value.
- The iterations are called as a sprints and at the end of each sprint you have the launch of a potentially deliverable software.

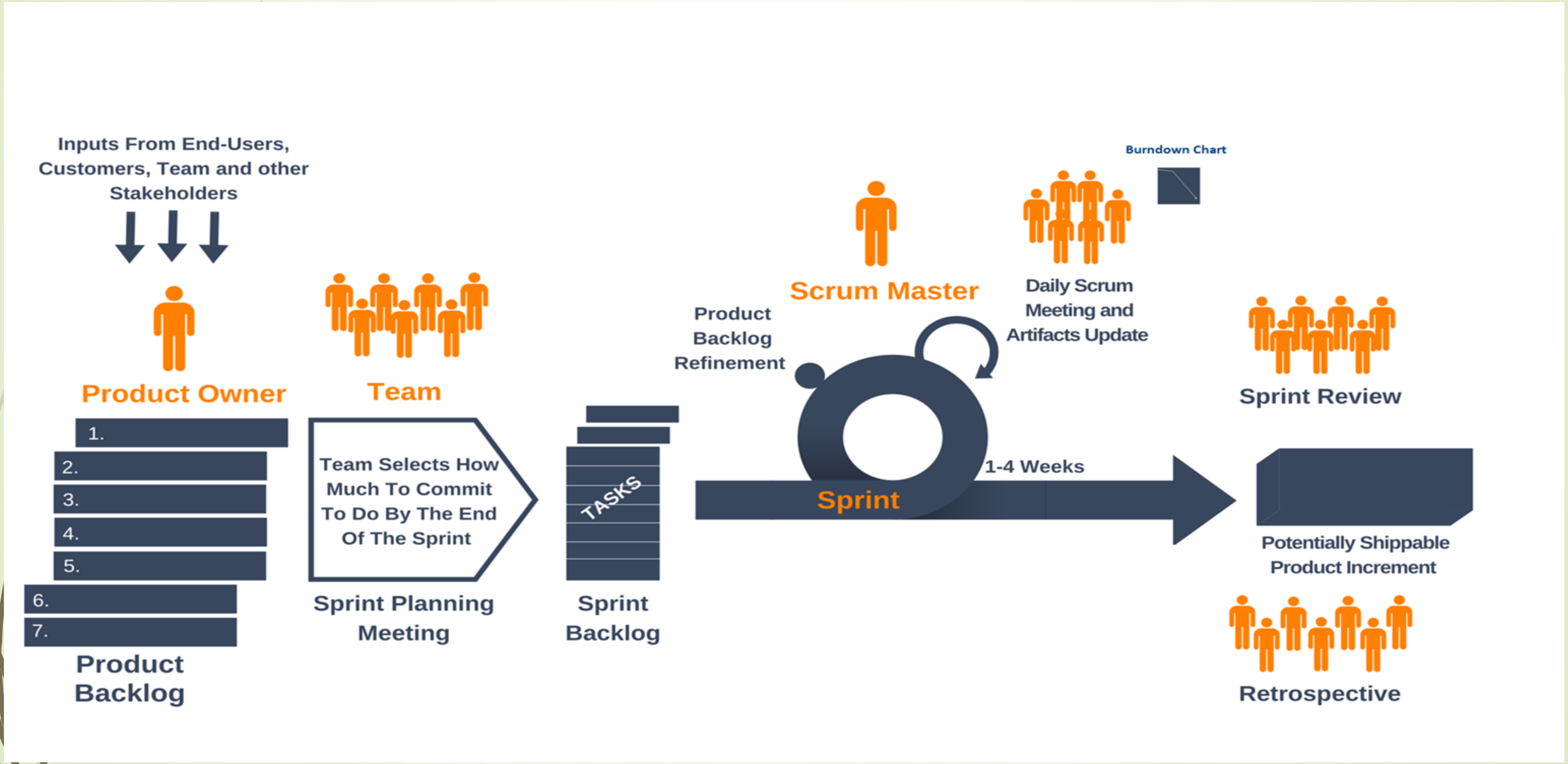
Scrum Model:

- **Product Owner:** The person with bright ideas who represents the customer's perspective.
- **Scrum Master:** The person who is implementing agile and preparing the team to follow the agile approach and making them more efficient (Project Manager).
  - The scrum master is not a supervisor.
  - The scrum master is not a secretary.
  - Self organization does not means the absence of management.
- **Team Member:** Team consist of cross-functional team members with expertise in areas such as design, developers, and testers.

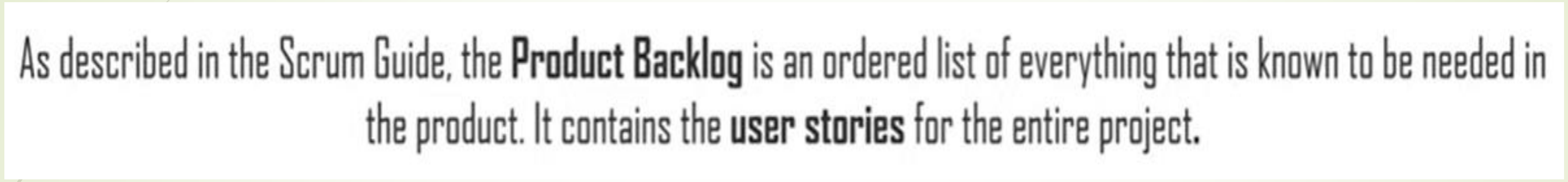
Scrum Lifecycle:



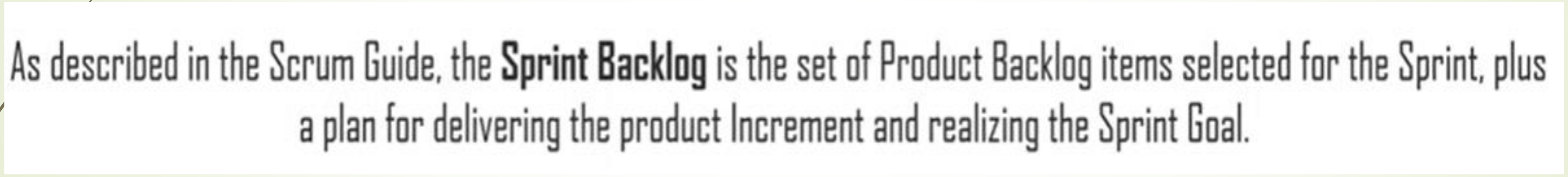
Scrum Lifecycle:


Inputs From End-Users, Customers, Team and other Stakeholders
Product Owner
Team
1.
2.
3.
4.
5.
6.
7.
Product Backlog
Team Selects How Much To Commit To Do By The End Of The Sprint
Sprint Planning Meeting
TASKS
Sprint Backlog
Product Backlog Refinement
Scrum Master
Daily Scrum Meeting and Artifacts Update
Burndown Chart
Sprint
1-4 Weeks
Sprint Review
Potentially Shippable Product Increment
Retrospective

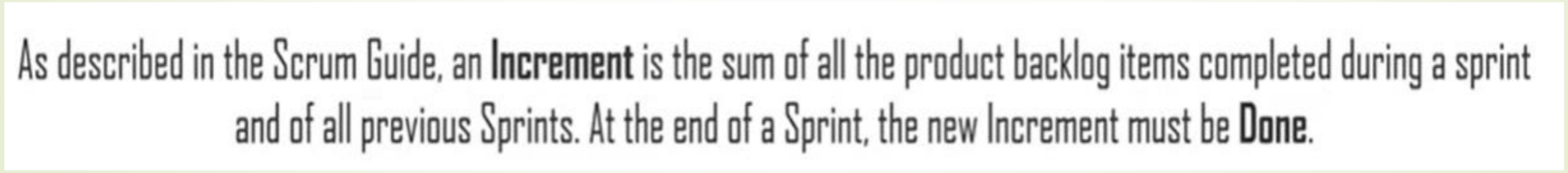
Scrum Lifecycle:



As described in the Scrum Guide, the **Product Backlog** is an ordered list of everything that is known to be needed in the product. It contains the **user stories** for the entire project.

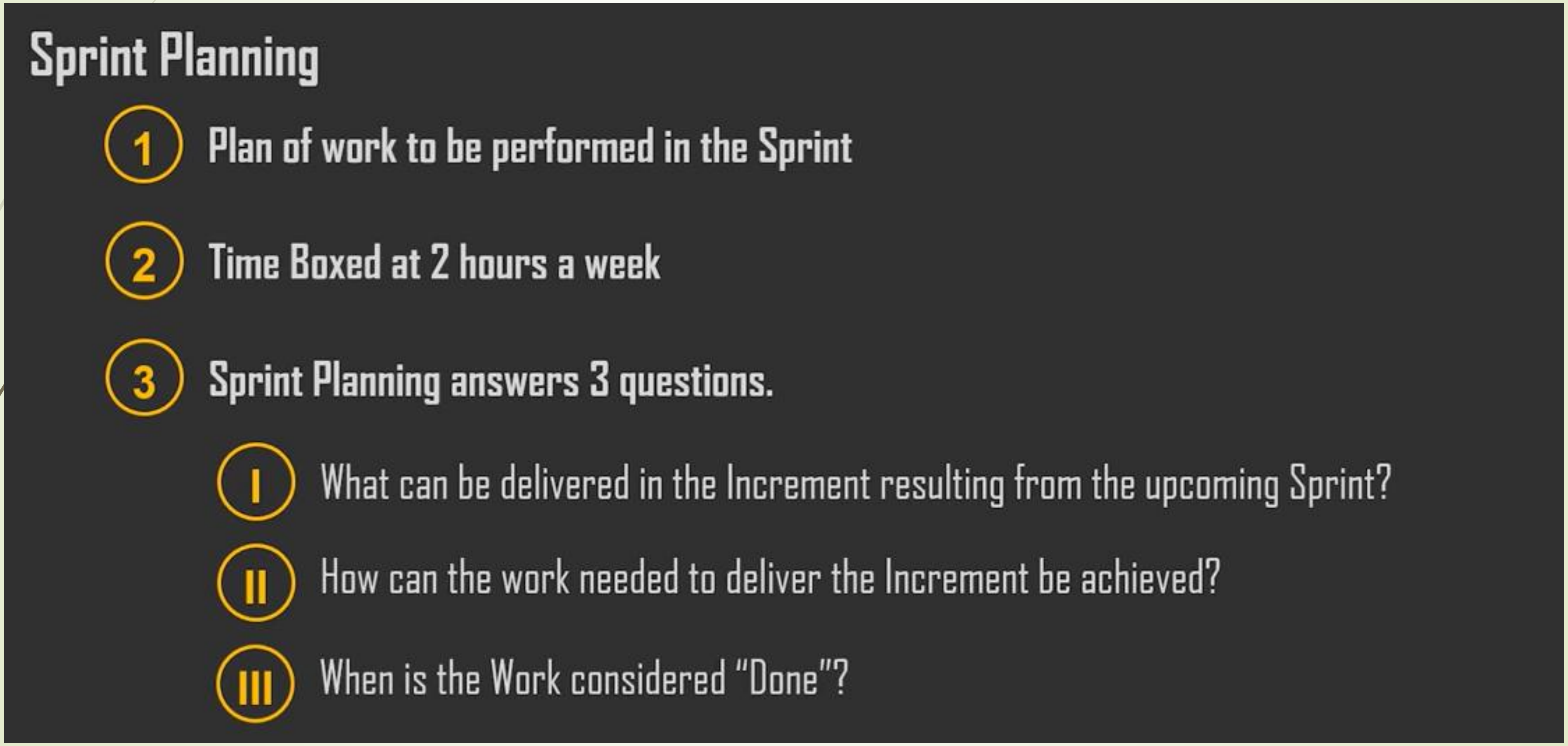


As described in the Scrum Guide, the **Sprint Backlog** is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal.



As described in the Scrum Guide, an **Increment** is the sum of all the product backlog items completed during a sprint and of all previous Sprints. At the end of a Sprint, the new Increment must be **Done**.

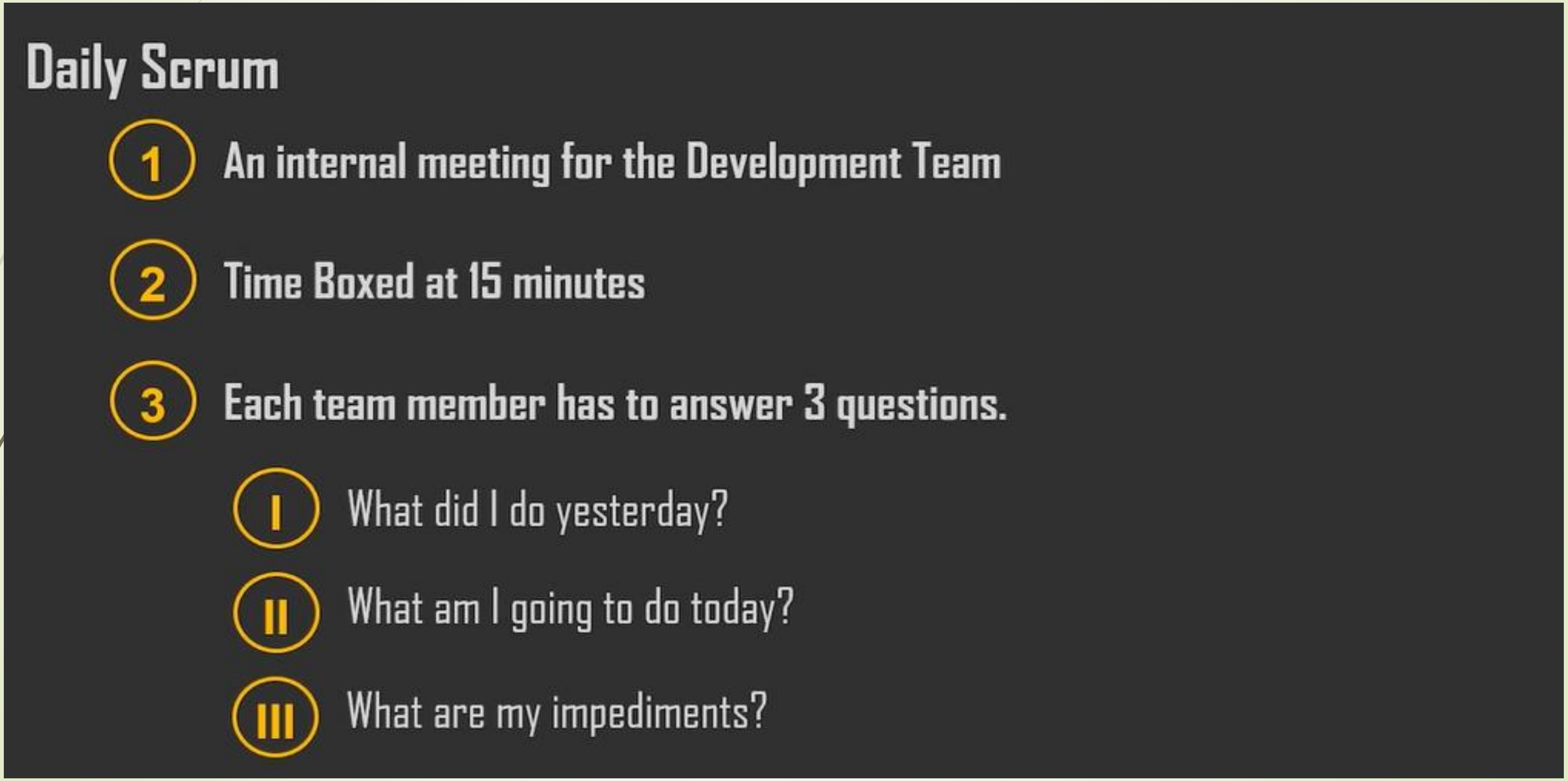
Scrum Lifecycle:



Sprint Planning

1. Plan of work to be performed in the Sprint
2. Time Boxed at 2 hours a week
3. Sprint Planning answers 3 questions.
   - I What can be delivered in the Increment resulting from the upcoming Sprint?
   - II How can the work needed to deliver the Increment be achieved?
   - III When is the Work considered "Done"?

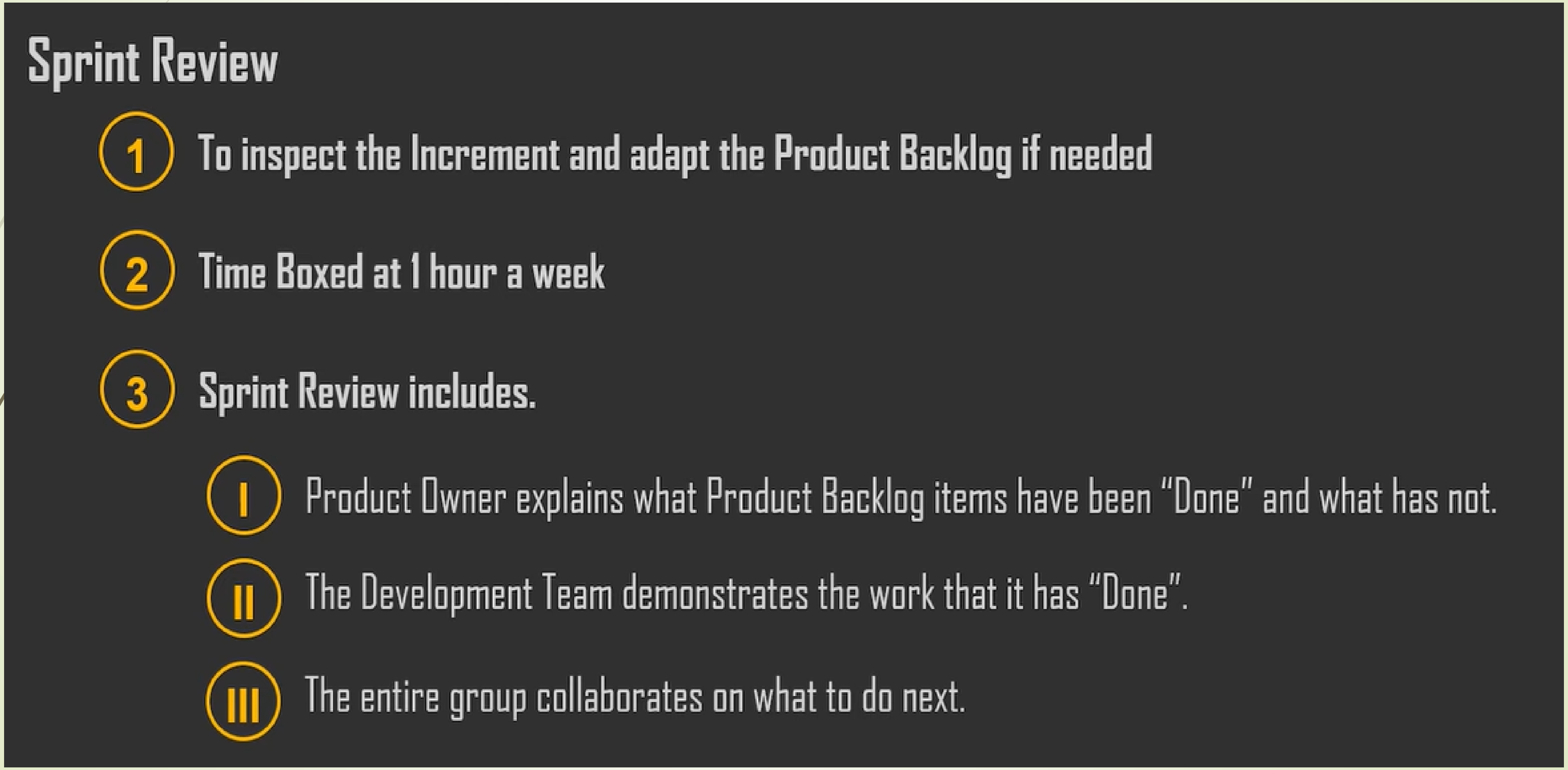
Scrum Lifecycle:



Daily Scrum

1. An internal meeting for the Development Team
2. Time Boxed at 15 minutes
3. Each team member has to answer 3 questions.
   - I What did I do yesterday?
   - II What am I going to do today?
   - III What are my impediments?

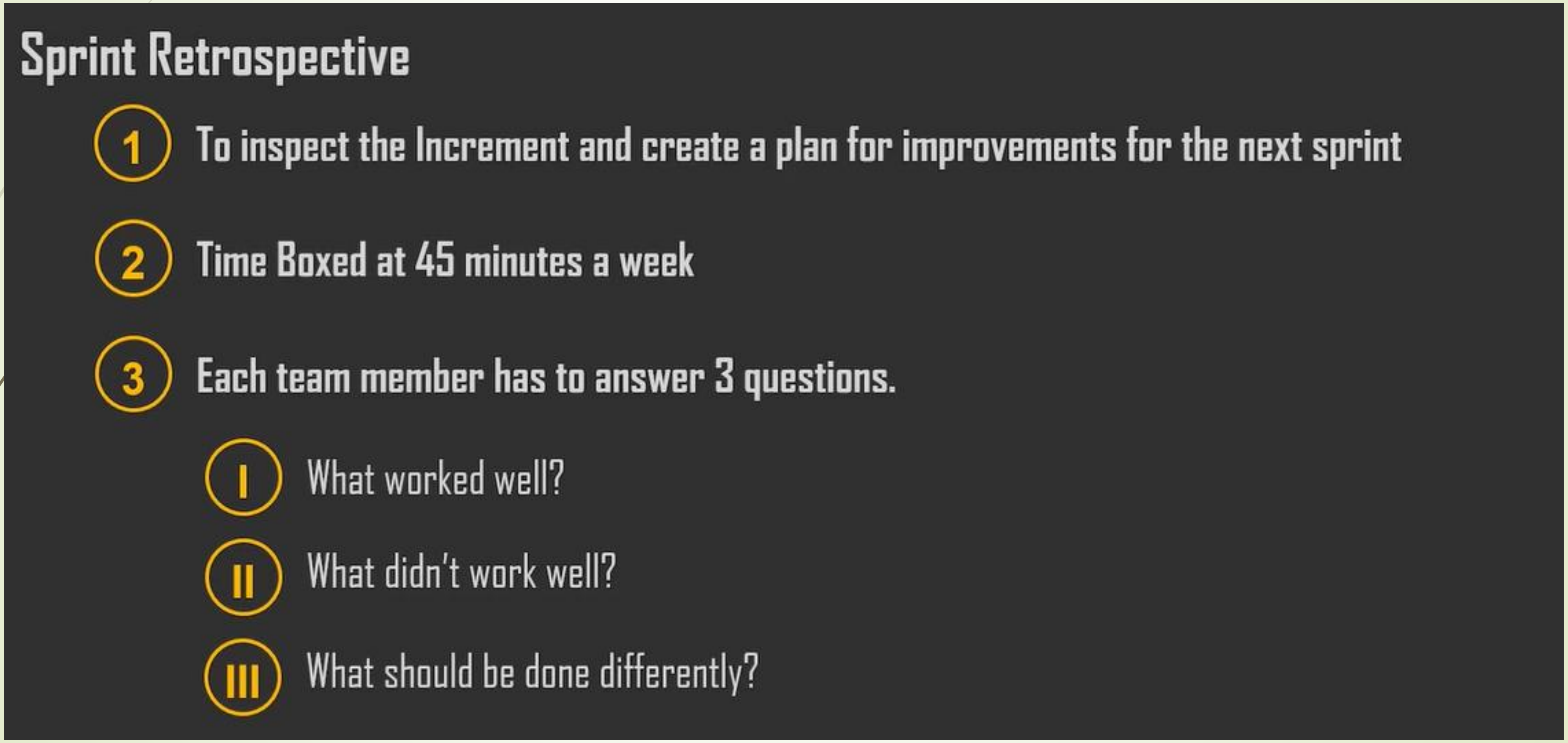
Scrum Lifecycle:



Sprint Review

1. To inspect the Increment and adapt the Product Backlog if needed
2. Time Boxed at 1 hour a week
3. Sprint Review includes.
   - I Product Owner explains what Product Backlog items have been "Done" and what has not.
   - II The Development Team demonstrates the work that it has "Done".
   - III The entire group collaborates on what to do next.

Scrum Lifecycle:



Sprint Retrospective

1. To inspect the Increment and create a plan for improvements for the next sprint
2. Time Boxed at 45 minutes a week
3. Each team member has to answer 3 questions.
   - I What worked well?
   - II What didn't work well?
   - III What should be done differently?

Agile Vs Structured Process Model

The Agile Process	**The Structured/Planned Process**
• Goal(s) - Comfort with change	• Goal(s) - Predictability and assurance to deal with mission criticality
• Tacit knowledge and communication	• Explicit, formal knowledge and communication
• Co-located, expert customer	• Contractual relationship with customer
• Small size skilled teams	• Large teams with a range of skills
• Dynamic team culture	• Comfort with stability
• Planning is a means to an end – but THERE IS LOTS OF PLANNING	• Process maturity and formal planning used to build trust
• Adds steps as needed	• Handles non-functional requirements better
	• Subtracts steps if appropriate

Agile vs Structured: Where do they work best?

- **Application**
  - Responding to change vs. high assurance and stability
  - Small teams vs. large teams
  - High-change, project-focused vs. Low-change and organization-focused
- **Management**
  - On site customers vs. As-needed interactions
  - Internalized plans and qualitative control vs. documented plans and quantitative control
  - Tacit knowledge and informal communication vs. explicit documented knowledge
- **Technical**
  - Informal, unforeseeable requirements, vs. foreseeable evolution
  - Simple design on known architecture vs. extensive design, and importance given to architecture
  - Executable test cases as requirements vs. Documented test plans
- **Personnel**
  - Experienced and co-located developers and customers vs. not always co-located
  - Thriving on dynamism vs. thriving on order

Agile vs. Structured e.g. Waterfall (contd)

- **the waterfall model**
  - 'One Phase at a time' and 'Rigid' development cycle
  - difficult to make last minute changes in requirements or design
  - suited for development of programs that are already stable.
    - where design does not need a major makeover OR
    - where the designers of a software can accurately predict the flaws that may arise
    - design is easier to manage and the development costs can be ascertained before hand.

Agile vs. Structured e.g. Waterfall (contd)

- **the agile models**
  - to be more efficient than the waterfall model, due to adaptability to the real world.
  - due to their iterative and adaptable nature, can incorporate changes and release a product in lesser time
  - more widely applicable than the waterfall model.
  - are applicable in every area of software development.
  - depends a lot more on the team effort of above average programmers, than relying on a few expert programmers.
  - best suited for web based applications where iterative nature helps in incorporating and correcting the various bugs that arise over time.

Selection of appropriate life cycle model for a project:

- Characteristics of the software to be developed
- Characteristics of the development team
- Risk associated with the project
- Characteristics of the customer

Thank You.