# Software Engineering (SE)

## Course Code: CS – 349

**Course Instructor:** Dr. Shitalkumar A Jain

# SE
# Course Objectives

- CS349.CEO.1: To identify the software process model
- CS349.CEO.2: To process requirement engineering for product development
- CS349.CEO.3: To learn design concepts and modelling for software development
- CS349.CEO.4: To comprehend the estimation and management of software metrics
- CS349.CEO.5: To Understand test driven environment in software development

# SE
# Course Outcomes

- CS349.CO.1: Classify process models
- CS349.CO.2: Analyze conformance of the requirement related to project development
- CS349.CO.3: Develop design models using UML diagram
- CS349.CO.4: Mitigate the risk associated with project development
- CS349.CO.5: Evaluate the schedule, cost and staff associated with project
- CS349.CO.6: Review quality assurance through test driven development

# UNIT – I
## Basics of Software Engineering

- **App/System/Case study:**

  Learning Game Design and Software Engineering through a Game Prototyping Experience

- **Content:**

  Generic process model: Process framework, umbrella activities, Process Adaptation, Perspective Process Models - Waterfall Model, Prototyping, Incremental, and Agile Process Model: XP and Scrum, introduction to Principles of framework Activities, DevOps concepts and process: continuous development, continuous integration, continuous testing, continuous deployment, and continuous monitoring.

- **Self-Study:** Component based development process model.
- **Further Reading:** Dynamic System Development Method

# WATERFALL MODEL

# Waterfall Model

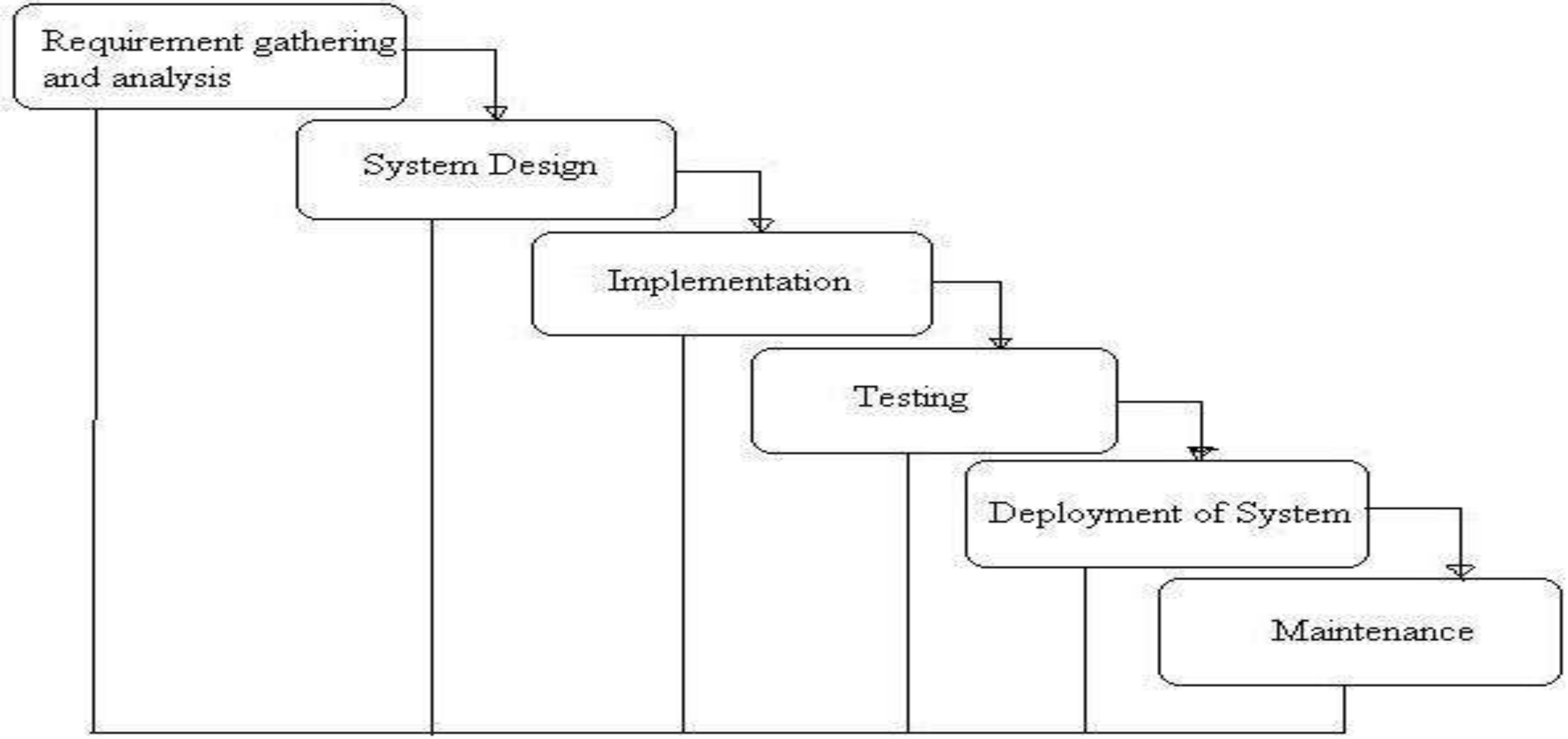
- It is also referred to as a linear-sequential life cycle model.
- It is very simple to understand and use.
- In a waterfall model, each phase must be completed fully before the next phase can begin
- Used for the project which is small and there are no uncertain requirements.
- At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project.

# Waterfall Model

- Used to develop enterprise applications like
  - Customer Relationship Management (CRM) systems
  - Human Resource Management Systems (HRMS)
  - Supply Chain Management Systems
  - Inventory Management Systems, etc
- Used significantly in the development of software till the year 2000.
- Even after the Agile manifesto was published in 2001, Waterfall model continued to be used by many organization till the last decade.

# Waterfall Model

General Overview of "Waterfall Model"



# Requirement Gathering and Analysis

- Requirements are gathered by the business analyst and analyzed by the team.
- Requirements are documented during this phase and clarifications can be sought. (**Deliverables – Software Requirements Specifications (SRS))**
- The Business Analysts document the requirement based on their discussion with the customer.

# System Design

- The architect and senior members of the team work on the software architecture, high level and low level design for the project.
- The architect creates the Architecture diagrams and high level / low level design documents. **(Deliverables – Design Documents)**

# Implementation

- The development team works on coding the project.
- They take the design documents / artifacts and ensure that their solution follows the design finalized by the architect
- They also perform several other activities like a senior developer reviewing the other developers code for any issues. **(Deliverables – Deployed Software)**
- Some developers perform static analysis of the code.

# Testing

- The testing team tests the complete application and identifies any defects in the application.
- These defects are fixed by the developers, and then the testing team tests the fixes to ensure that the defects are fixed
- They also perform regression testing of the application to see if any new defects were introduced. **(Deliverables – Quality Assured)**

# Deployment

- The team builds and installs the application on the servers which were procured for the application.
- Some of the high level activities includes
  - installing the OS on the servers
  - installing security patches
  - hardening the servers
  - installing web servers and application servers
  - installing the database etc.

# Maintenance

- During the maintenance phase, the team ensures that the application is running smoothly on the servers without any downtime.
- Issues that are reported after going live are fixed by the team and tested by the testing team.

# Advantages of waterfall model

- This model is simple, easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are clearly defined and very well understood.

# Disadvantages of waterfall model

- Once an application is in the **testing** stage, it is very difficult to go back and change something that was not well-thought out in the concept stage
- No working software is produced until late during the life cycle
- High amounts of risk and uncertainty
- Not a good model for complex and object-oriented projects
- Poor model for long and ongoing projects
- Not suitable for the projects where requirements are at a moderate to high risk of changing

# When to use the waterfall model

- Can be used when the requirements are very well known, clear and fixed
- Product definition is stable
- Technology is understood
- There are no ambiguous requirements
- Ample resources with required expertise are available freely
- The project is short

# PROTOTYPE MODEL

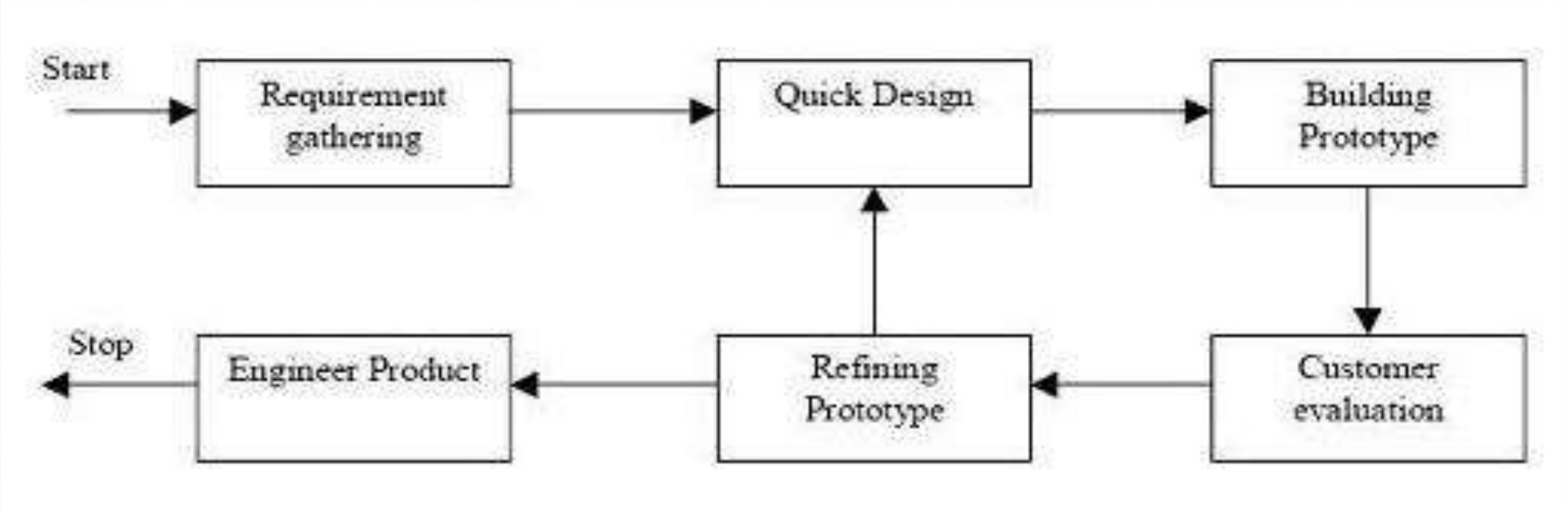
# Prototype model

- Prototype model is a software development model
- Instead of freezing the requirements before, a design or coding can proceed, and a throwaway prototype is built to understand the requirements
- Prototype are developed based on the currently known requirements
- By using prototype, the client can get an “actual feel” of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system

# Prototype model

- Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements
- The prototype are usually not complete systems and many of the details are not built in the prototype
- The goal is to provide a system with overall functionality

# Prototype Model



*Prototyping Model*

# Advantages of Prototype model

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed
- Errors can be detected much earlier
- Quicker user feedback is available leading to better solutions
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified.

# Disadvantages of Prototype model

- Leads to implementing and then repairing way of building systems
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans
- Incomplete application may cause application not to be used as the full system was designed Incomplete or inadequate problem analysis

# When to use Prototype model

- Prototype model should be used when the desired system needs to have **a lot of interaction** with the end users

- 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 usable system.**

- They are excellent for designing good **human computer interface** systems.

# INCREMENTAL MODEL

# Incremental Model

- In incremental model the whole requirement is divided into various builds.

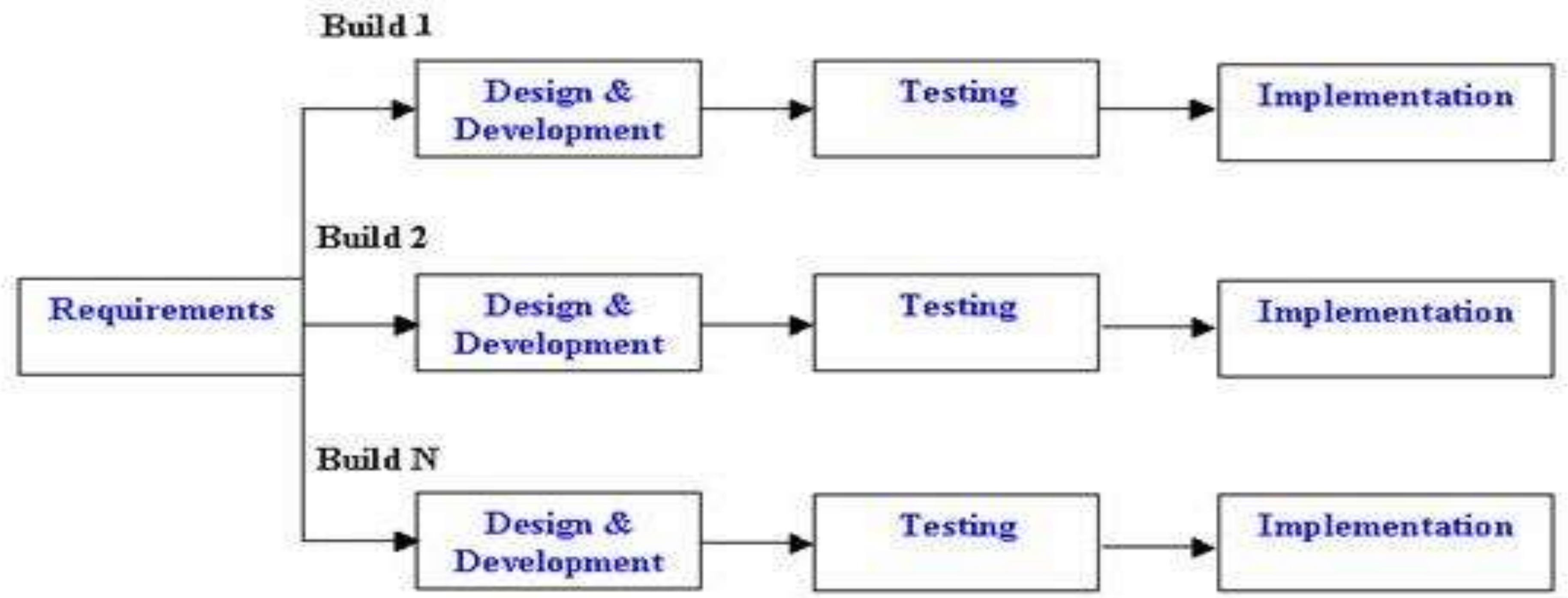
- Multiple development cycles take place here, making the life cycle a **“multi-waterfall” cycle**.

- Cycles are divided up into smaller, more easily managed modules.

# Incremental Model - Working

- Each module passes through the requirements, design, implementation and **testing** phases.
- A working version of software is produced during the first module, to have working software early on during the **software life cycle**.
- Each subsequent release of the module adds function to the previous release.
- The process continues till the complete system is achieved.

# Incremental Model - Working



Incremental Life Cycle Model

# Advantages of Incremental model

- Generates working software quickly and early during the software life cycle
- This model is more flexible – less costly to change scope and requirements
- It is easier to test and debug during a smaller iteration
- In this model the customer can respond to each built
- Lowers initial delivery cost
- Easier to manage risk because risky pieces are identified and handled during it's iteration

# Disadvantages of Incremental model

- Needs good planning and design
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally
- Total cost is higher than **waterfall model**

# When to use the Incremental model

- When the requirements of the complete system are clearly defined and understood
- Major requirements must be defined; however, some details can evolve with time
- There is a need to get a product to the market early
- A new technology is being used
- Resources with needed skill set are not available
- There are some high risk features and goals

# SPIRAL MODEL

# Spiral model

- The spiral model is similar to the incremental model, with more emphasis placed on risk analysis
- The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation
- A software project repeatedly passes through these phases in iterations

# Planning Phase

- Requirements are gathered during the planning phase.
- Requirements like ‘BRS’ that is ‘Business Requirement Specifications’ and ‘SRS’ that is ‘System Requirement specifications’.

# Risk Analysis

- In the **risk analysis phase**, a process is undertaken to identify risk and alternate solutions
- A prototype is produced at the end of the risk analysis phase
- If any risk is found during the risk analysis then alternate solutions are suggested and implemented

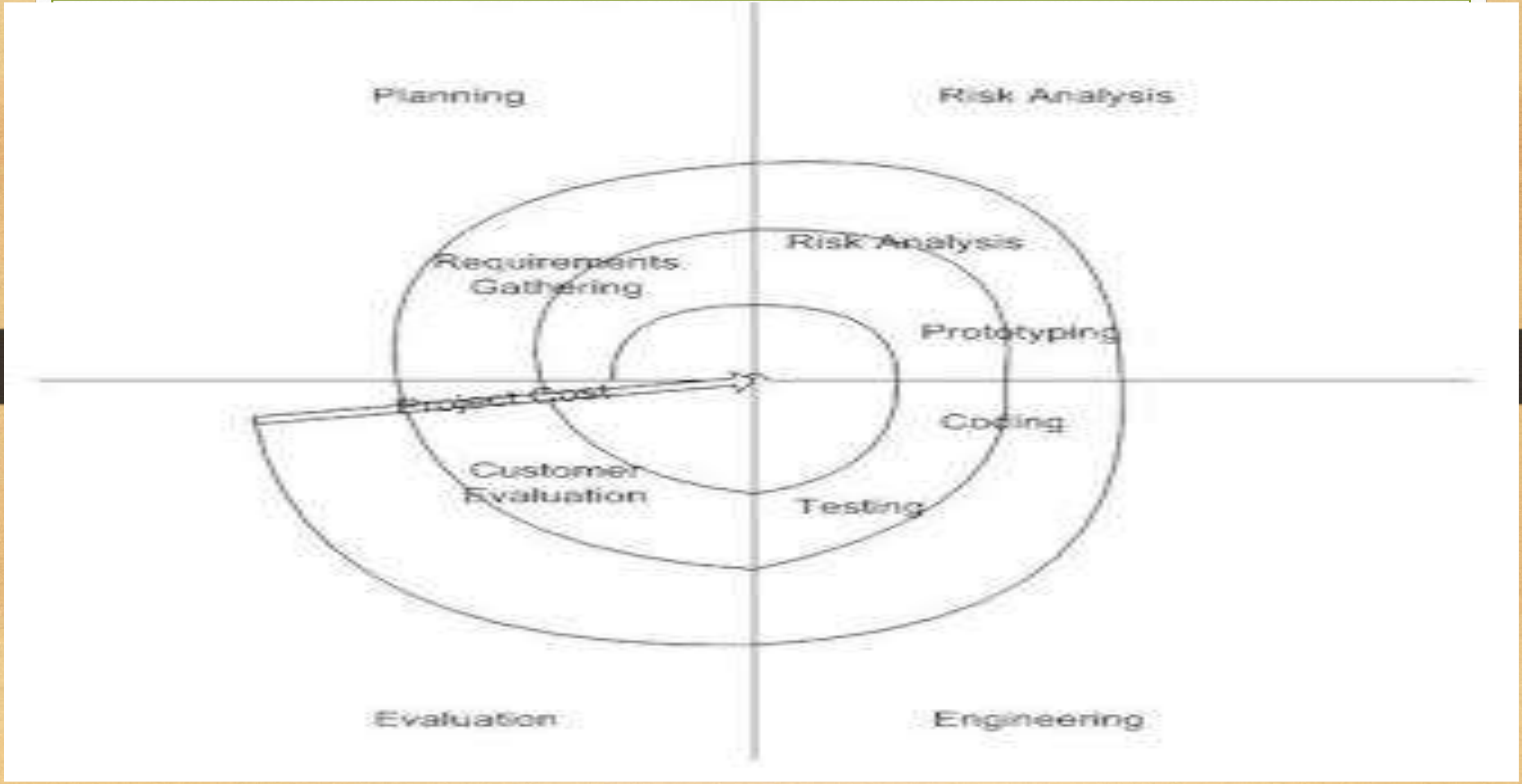
# Engineering Phase

- In this phase software is **developed**, along with **testing** at the end of the phase.
- Hence in this phase the development and testing is done.

# Evaluation phase

- This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

# Spiral Model



# Advantages of Spiral model

- High amount of risk analysis hence, avoidance of Risk is enhanced
- Good for large and mission-critical projects
- Strong approval and documentation control
- Additional Functionality can be added at a later date
- Software is produced early in the **software life cycle**

# Disadvantages of Spiral model

- Can be a costly model to use
- Risk analysis requires highly specific expertise
- Project's success is highly dependent on the risk analysis phase
- Doesn't work well for smaller projects

# When to use Spiral model

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

# RAD MODEL

# RAD Model

- RAD model is Rapid Application Development model
- It is a type of **incremental model**
- In RAD model the 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

# RAD Model

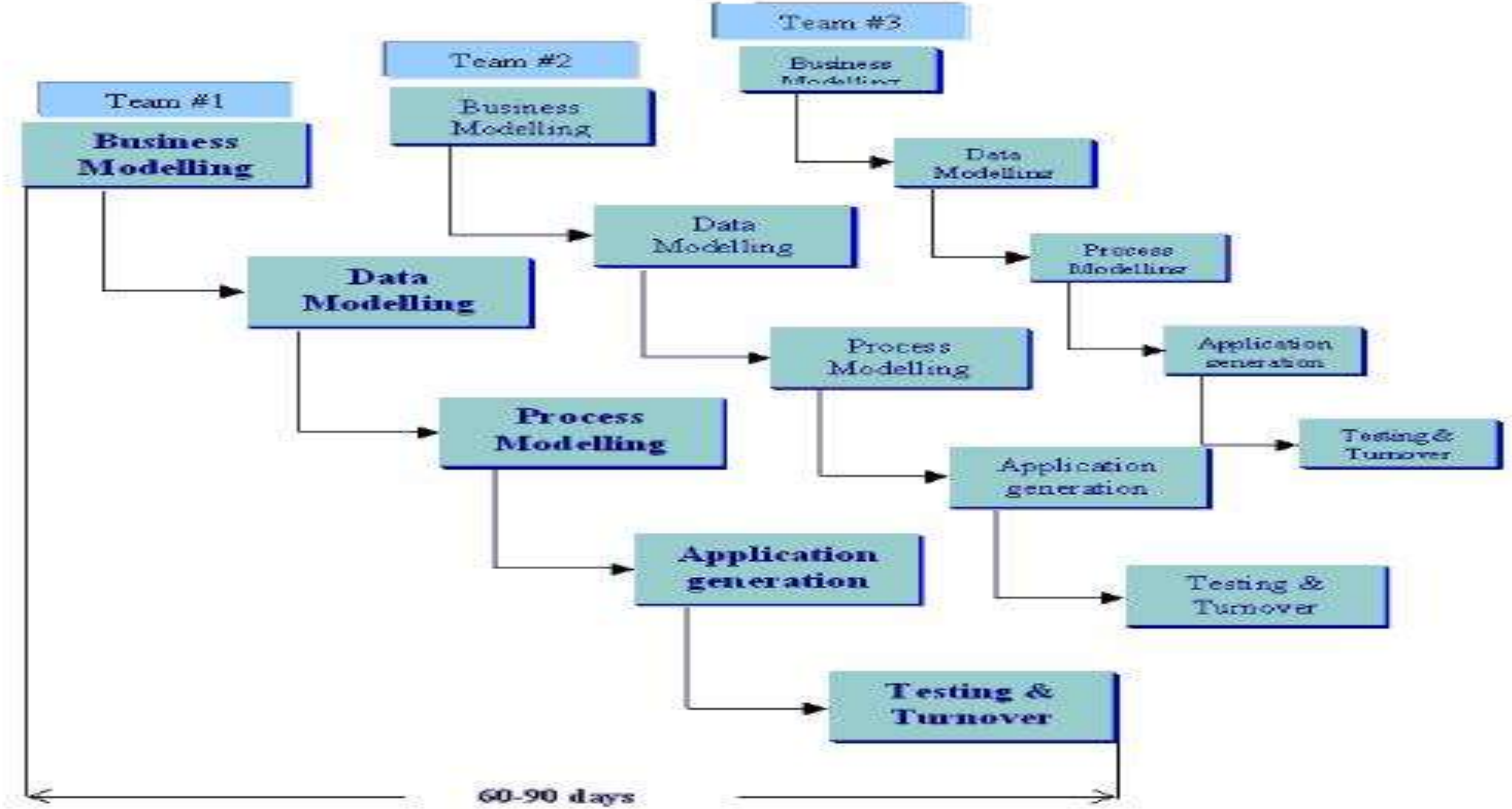


Figure 1.5 – RAD Model

# RAD Model - Phases

- **Business modeling:** The information flow is identified between various business functions.
- **Data modeling:** Information gathered from business modeling is used to define data objects that are needed for the business
- **Process modeling:** Data objects defined in data modeling are converted to achieve the business information flow, to achieve some specific business objective
- **Application generation:** Automated tools are used to convert process models into code and the actual system
- **Testing and turnover:** Test new components and all the interfaces

# Advantages of the RAD model

- Reduced development time
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of **integration issues**

# Disadvantages of RAD model

- Depends on strong team and individual performances for identifying business requirements
- Only system that can be modularized can be built using RAD
- Requires highly skilled developers/designers
- High dependency on modeling skills
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high

# When to use RAD model

- Used when there is a need to create a system that can be modularized in 2-3 months of time.
- Used if there's high availability of designers for modeling and the budget is high enough to afford their cost
- If resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months)

# Software Engineering, Testing and Quality Assurance

## UNIT - I

Shitalkumar A Jain

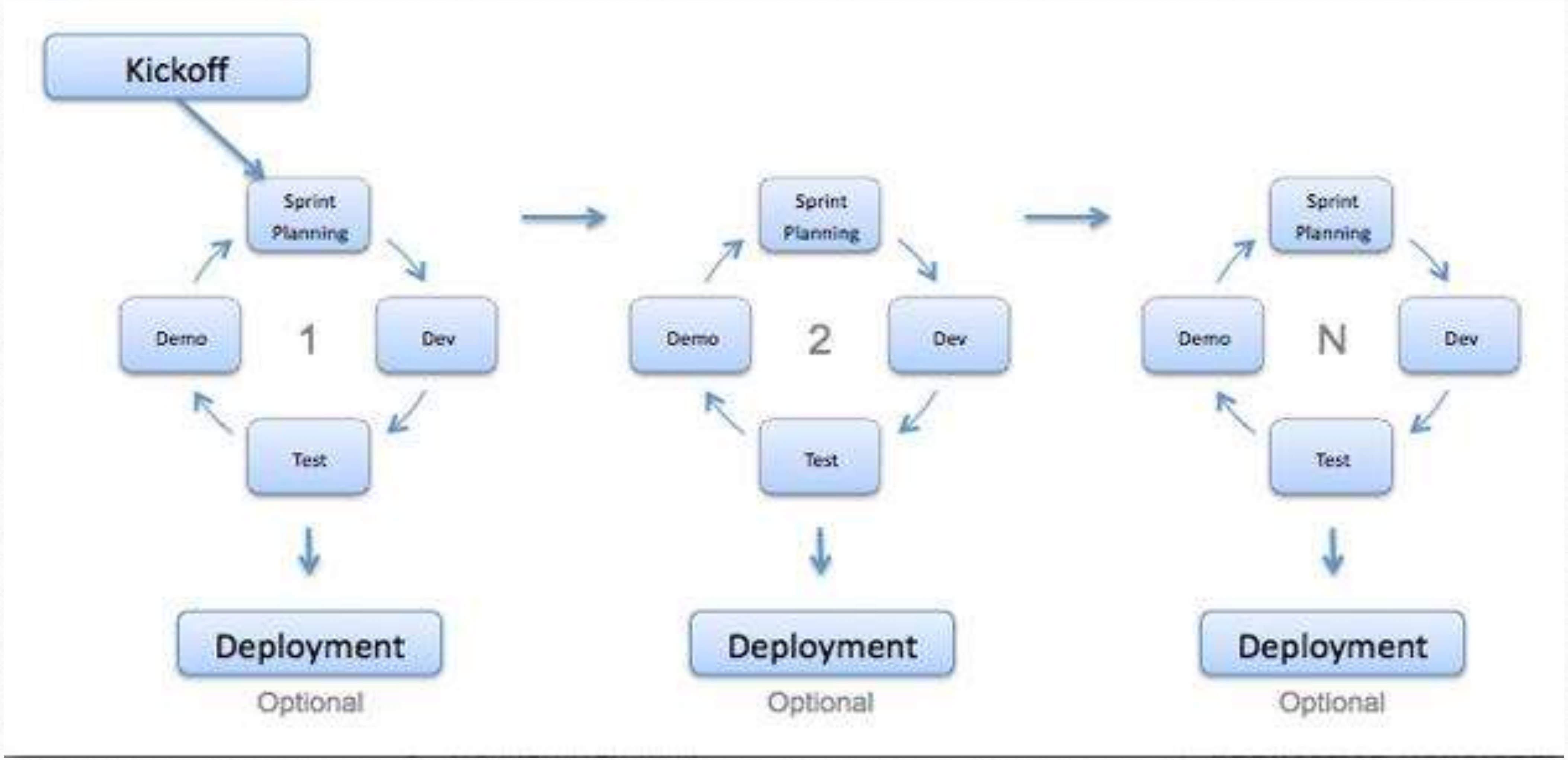
# UNIT – I
# Topics

- Process Models
  - Waterfall Model
  - Prototyping
  - Incremental
  - Spiral
  - RAD
- Agile methods of software development
  - Extreme Programming (XP)
  - Scrum
  - Cleanroom Approach

# Agile methods of software development

- Agile development model is also a type of Incremental model.
- Software is developed in incremental, rapid cycles.
- This results in small incremental releases with each release building on previous functionality.
- Each release is thoroughly tested to ensure software quality is maintained.
- It is used for time critical applications.
- Extreme Programming (XP) is currently one of the most well known agile development life cycle models.

# Agile Model


Kickoff
Sprint Planning
Demo
1
Dev
Test
Deployment
Optional
Sprint Planning
Demo
2
Dev
Test
Deployment
Optional
Sprint Planning
Demo
N
Dev
Test
Deployment
Optional

# Advantages of Agile model

- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

# Disadvantages of Agile model

- In case of some software deliverables, especially the large ones, it is **difficult to assess the effort required at the beginning** of the software development life cycle.
- There is **lack of emphasis on necessary designing and documentation**.
- The project can easily get taken off the track if the **customer representative is not clear** what final outcome that they want.
- Only **senior programmers are capable of taking the kind of decisions** required during the development process.
- Hence it has **no place for newbie programmers**, unless combined with experienced resources.

# When to use Agile model

- When new changes are needed to be implemented. The freedom agile gives to change is very important.
- New changes can be implemented at very little cost because of the frequency of new increments that are produced.
- To implement a new feature, the developers need to lose only the work of a few days, or even hours, to roll back and implement it.
- Unlike the waterfall model, in agile model very limited planning is required to get started with the project.
- Agile assumes that the end users' needs are ever changing in a dynamic business and the IT world.

# When to use Agile model

- Changes can be discussed and features can be newly added or removed based on feedback
- This effectively gives the customer the finished system they want or need
- Both system developers and stakeholders finds that they also get more freedom of time and options, than if the software was developed in a more rigid sequential way
- Having options gives them the ability to leave important decisions until more or better data or even entire hosting programs are available

# Extreme Programming

# Extreme Programming (XP)

- Extreme programming (XP) is one of the most important software development frameworks of Agile models.
- It is used to improve software quality, and it is responsive to customer requirements.
- The extreme programming model recommends taking the best practices that have worked well in the past in program development projects to extreme levels.

# Good practices needs to practiced extreme programming

- **Code Review**
  - Code review detects and corrects errors efficiently.
  - It suggests pair programming as coding and reviewing of written code carried out by a pair of programmers who switch their works between them **every hour**.
- **Testing**
  - Testing code helps to remove errors and improves its reliability.
  - XP suggests test-driven development (TDD) to continually write and execute test cases.
  - In the TDD approach, test cases are written even before any code is written.

# Good practices needs to practiced extreme programming

- **Incremental development**
  - Incremental development is very good because customer feedback is gained in early stage.
  - Based on this development team come up with new increments every few days after each iteration.
- **Simplicity**
  - Simplicity makes it easier to develop good quality code as well as to test and debug it.

# Good practices needs to practiced extreme programming

- **Design**
  - Good quality design is important to develop a good quality software. So, everybody should design **daily.**
- **Integration testing**
  - It helps to identify bugs at the interfaces of different functionalities.
  - Extreme programming suggests that the developers should achieve continuous integration by building and performing integration testing **several times a day**.

# Basic principles of Extreme programming

- XP is based on the frequent iteration through which the developers implement **User Stories.**
- User stories are simple and informal statements of the customer about the functionality needed.
- A User story is a conventional description by the user about a feature of the required system.
- It does not mention finer details such as the different scenarios that can occur.

# Basic principles of Extreme programming

- On the basis of User stories, the project team proposes Metaphors.
- Metaphors are a common vision of how the system would work.
- The development team may decide to build a Spike for some feature.
- A Spike is a very simple program that is constructed to explore the suitability of a solution being proposed.
- It can be considered similar to a prototype.

# Basic activities followed in software development : XP model

- **Coding**
  - The concept of coding which is used in XP model is slightly different from traditional coding.
  - Coding activity includes
    - drawing diagrams (modeling) that will be transformed into code
    - scripting a web-based system and
    - choosing among several alternative solutions
- **Testing**
  - XP model gives high importance on testing and considers it to be the primary factor to develop a fault-free software.

# Basic activities followed in software development : XP model

- **Listening**
  - The developers needs to carefully listen to the customers if they have to develop a good quality software.
  - Sometimes programmers may not have the in- depth knowledge of the system to be developed.
  - It is desirable for the programmers to properly understand the functionality of the system and they have to listen to the customers.
- **Designing**
  - Without a proper design, a system implementation becomes too complex and very difficult to understand the solution. Thus it makes maintenance expensive.
  - A good design results elimination of complex dependencies within a system. So, effective use of suitable design is emphasized.

# Basic activities followed in software development : XP model

- **Feedback**
  - One of the most important aspects of the XP model is to gain feedback to understand the exact customer needs
  - Frequent contact with the customer makes the development effective
- **Simplicity**
  - The main principle of the XP model is to develop a simple system that will work efficiently in present time, rather than trying to build something that would take time and it may never be used.
  - It focuses on some specific features that are immediately needed, rather than engaging time and effort on speculations of future requirements.

# XP model

- **Applications of Extreme Programming (XP)**
  - Some of the projects that are suitable to develop using XP model are given below:
- **Small projects**
  - XP model is very useful in small projects consisting of small teams as face to face meetings is easier to achieve.
- **Projects involving new technology or Research projects**
  - This type of projects face changing requirements rapidly and technical problems. So XP model is used to complete this type of projects.

# RUP (Rational Unified Process)

# RUP (Rational Unified Process)

- The RUP recognizes that conventional process models present a single view of the process.
- In contrast, the RUP is normally described from three perspectives:
  - A dynamic perspective, which shows the phases of the model over time.
  - A static perspective, which shows the process activities that are passed.
  - A practice perspective, which suggests good practices to be used during the process.

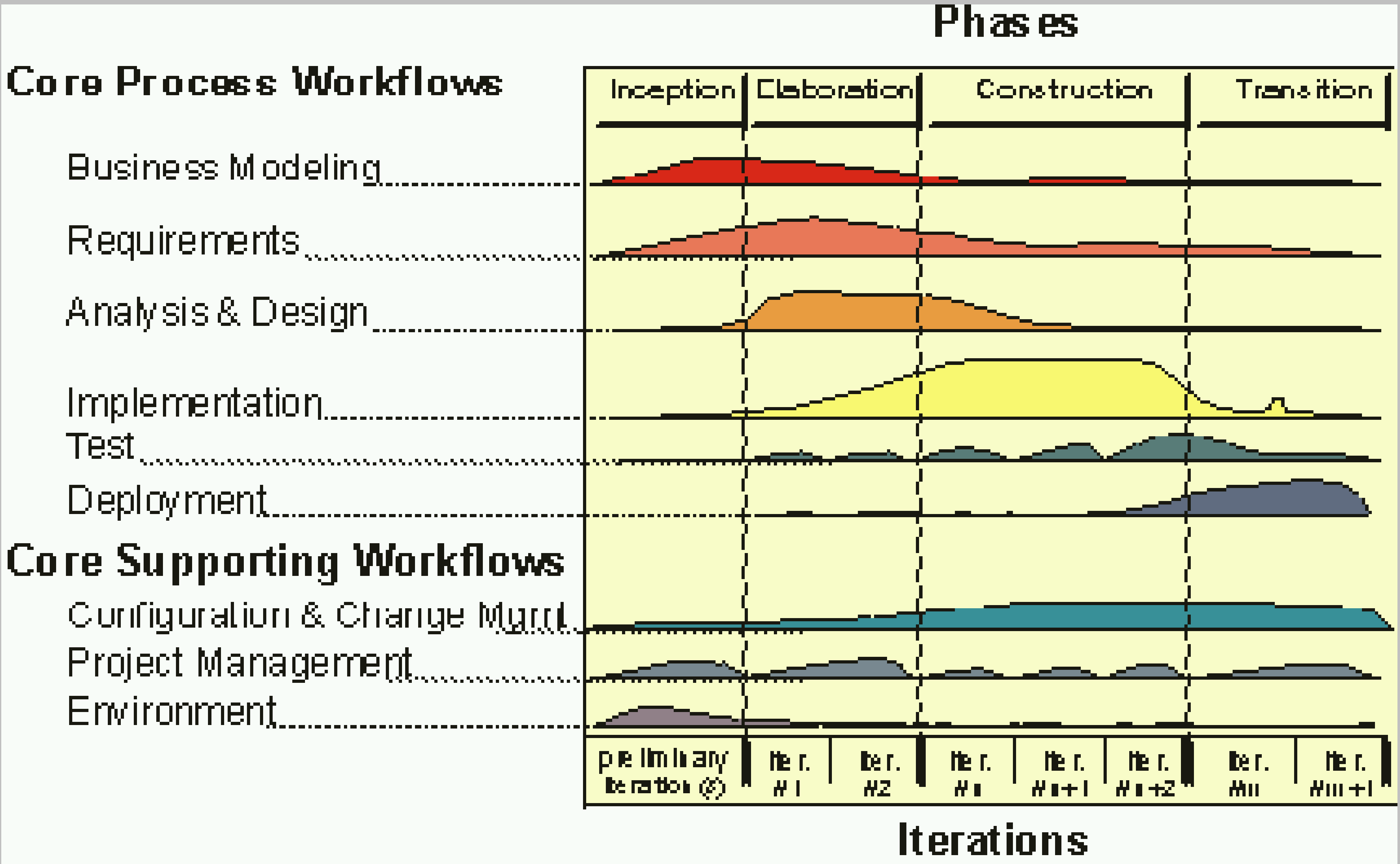
# The Rational Unified Process

- RUP is a method of managing OO Software Development
- It can be viewed as a Software Development Framework which is extensible and features are:
  - Iterative Development
  - Requirements Management
  - Component-Based Architectural Vision
  - Visual Modeling of Systems
  - Quality Management
  - Change Control Management

| WORKFLOW | DESCRIPTION |
|---|---|
| **Business modelling** | The business processes are modelled using business use cases. |
| **Requirements** | Actors who interact with the system are identified and use cases are developed to model the system requirements. |
| **Analysis and design** | A design model is created and documented using architectural models; component models; object models and sequence models. |
| **Implementation** | The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process. |
| **Testing** | Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation. |
| **Deployment** | A product release is created; distributed to users and installed in their workplace. |
| **Configuration and change management** | This supporting workflow managed changes to the system |
| **Project management** | This supporting workflow manages the system development (see Chapters 22 and 23). |
| **Environment** | This workflow is concerned with making appropriate software tools available to the software development team. |

# RUP Features

- Online Repository of Process Information and Description in HTML format
- Templates for all major artifacts, including:
  - RequisitePro templates (requirements tracking)
  - Word Templates for Use Cases
  - Project Templates for Project Management
- Process Manuals describing key processes


Phases
Core Process Workflows
Inception
Elaboration
Construction
Transition
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Core Supporting Workflows
Configuration & Change Mgmt
Project Management
Environment
preliminary iteration(s)
iter. #1
iter. #2
iter. #n
iter. #n+1
iter. #n+2
iter. #m
iter. #m+1
Iterations

# An Iterative Development Process

- Recognizes the reality of changing requirements
  - Caspers Jones's research on 8000 projects
    - 40% of final requirements arrived after the analysis phase, after development had already begun
- Promotes early risk mitigation, by breaking down the system into mini-projects and focusing on the riskier elements first
- Allows you to "plan a little, design a little, and code a little"
- Encourages all participants, including testers, integrators, and technical writers to be involved earlier on
- Allows the process itself to modulate with each iteration, allowing you to correct errors sooner and put into practice lessons learned in the prior iteration
- Focuses on component architectures, not final big bang deployments

# An Incremental Development Process

- Allows for software to evolve, not be produced in one huge effort
- Allows software to improve, by giving enough time to the evolutionary process itself
- Forces attention on stability, for only a stable foundation can support multiple additions
- Allows the system (a small subset of it) to actually run much sooner than with other processes
- Allows interim progress to continue through the stubbing of functionality
- Allows for the management of risk, by exposing problems earlier on in the development process

# Goals and Features of Each Iteration

- The primary goal of each iteration is to slowly chip away at the risk facing the project, namely:
  - performance risks
  - integration risks (different vendors, tools, etc.)
  - conceptual risks (ferret out analysis and design flaws)
- Perform a “miniwaterfall” project that ends with a delivery of something tangible in code, available for scrutiny by the interested parties, which produces validation or correctives
- Each iteration is risk-driven
- The result of a single iteration is an increment--an incremental improvement of the system, yielding an evolutionary approach

# Risk Management

- Identification of the risks
- Iterative/Incremental Development
- The prototype or pilot project
- Early testing and deployment as opposed to late testing in traditional methods

# The Development Phases

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

# Inception Phase

- Overriding goal is obtaining buy-in from all interested parties
- Initial requirements capture
- Cost Benefit Analysis
- Initial Risk Analysis
- Project scope definition
- Defining a candidate architecture
- Development of a disposable prototype
- Initial Use Case Model (10% - 20% complete)
- First pass at a Domain Model

# Elaboration Phase

- Requirements Analysis and Capture
  - Use Case Analysis
    - Use Case (80% written and reviewed by end of phase)
    - Use Case Model (80% done)
    - Scenarios
      - Sequence and Collaboration Diagrams
      - Class, Activity, Component, State Diagrams
  - Glossary (so users and developers can speak common vocabulary)
  - Domain Model
    - to understand the problem: the system's requirements as they exist within the context of the problem domain
  - Risk Assessment Plan revised
  - Architecture Document

# Construction Phase

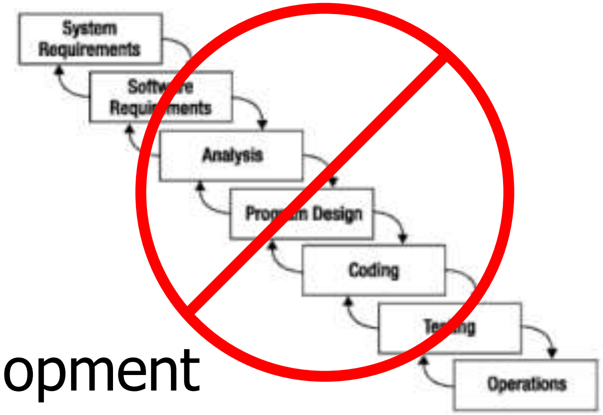
- Focus is on implementation of the design:
  - cumulative increase in functionality
  - greater depth of implementation (stubs fleshed out)
  - greater stability begins to appear
  - implement all details, not only those of central architectural value
  - analysis continues, but design and coding predominate

# Transition Phase

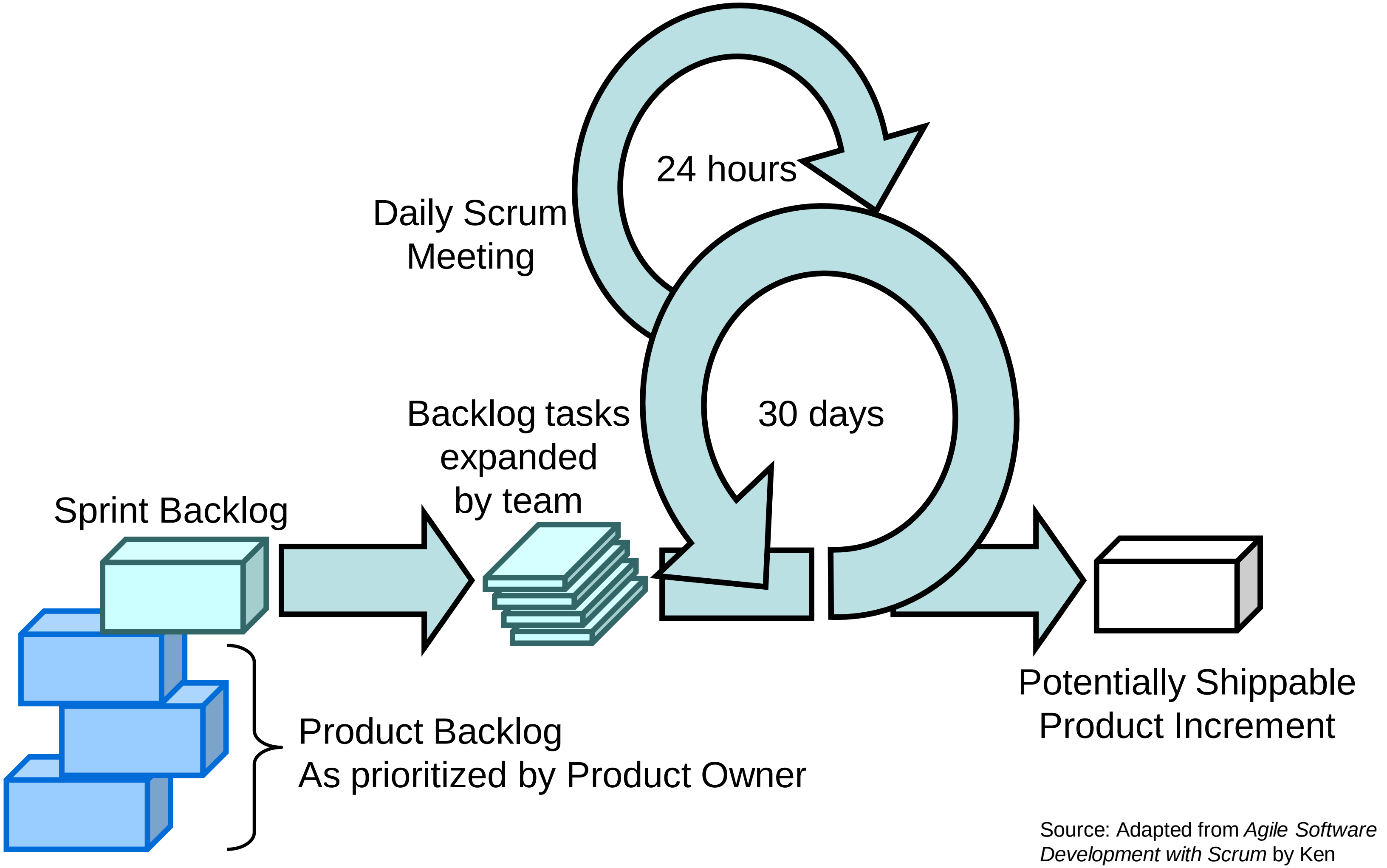
- The transition phase consists of the transfer of the system to the user community
- It includes manufacturing, shipping, installation, training, technical support and maintenance
- Development team begins to shrink
- Control is moved to maintenance team
- Alpha, Beta, and final releases
- Software updates
- Integration with existing systems (legacy, existing versions, etc.)

# What is Scrum?

- **Scrum**: **It's about common sense**
  - Is an agile, lightweight process
  - Can manage and control software and product development
  - Uses iterative, incremental practices
  - Has a simple implementation
  - Increases productivity
  - Reduces time to benefits
  - Embraces adaptive, empirical systems development
  - Is not restricted to software development projects
  - Embraces the opposite of the waterfall approach…

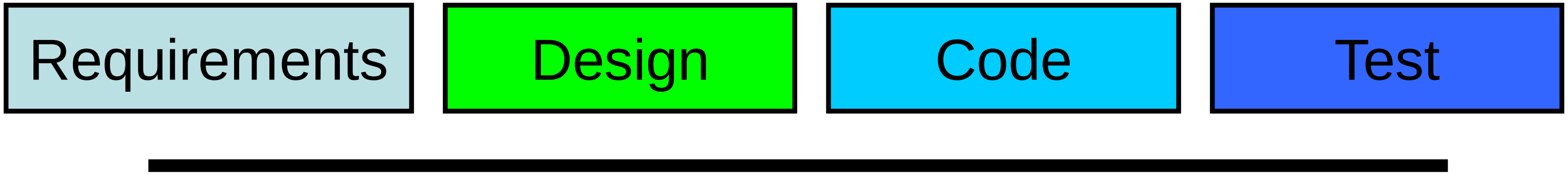


# Scrum at a Glance


24 hours
Daily Scrum Meeting
Backlog tasks expanded by team
30 days
Sprint Backlog
Potentially Shippable Product Increment
Product Backlog As prioritized by Product Owner

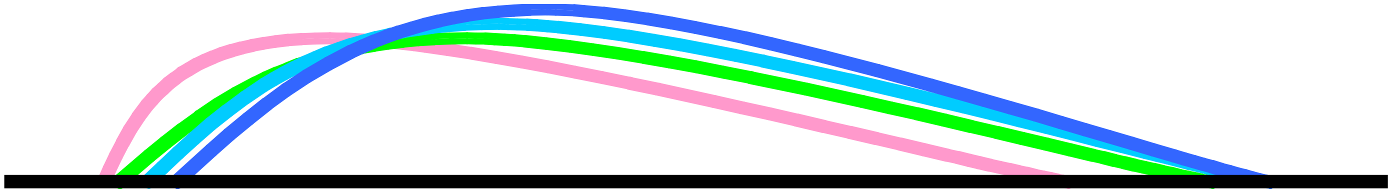
Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

# Sequential vs. Overlap



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

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



# 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

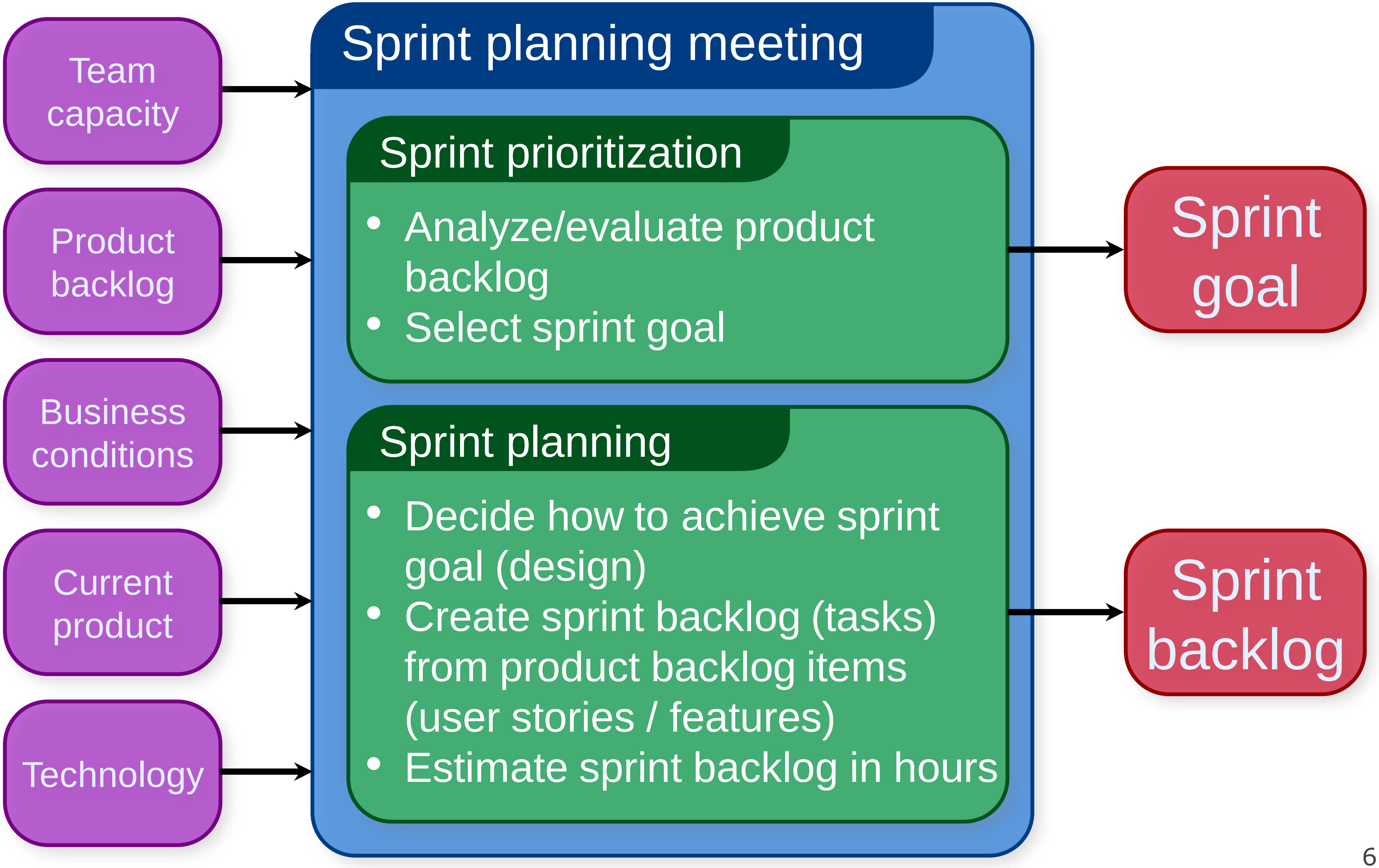
# Scrum Roles

- 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 Planning Mtg.



Team capacity

Product backlog

Business conditions

Current product

Technology

## Sprint planning meeting

### Sprint prioritization

- Analyze/evaluate product backlog
- Select sprint goal

### Sprint planning

- Decide how to achieve sprint goal (design)
- Create sprint backlog (tasks) from product backlog items (user stories / features)
- Estimate sprint backlog in hours

Sprint goal

Sprint backlog

# Daily Scrum Meeting

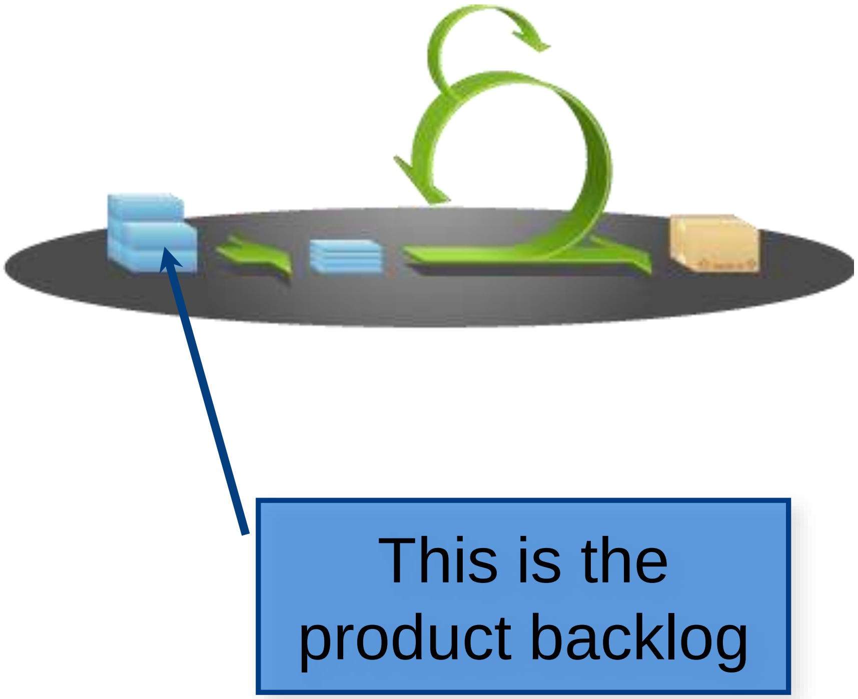
- Parameters
  - Daily, ~15 minutes, Stand-up
  - Anyone late pays a $1 fee
- Not for problem solving
  - Whole world is invited
  - Only team members, Scrum Master, product owner, can talk
  - Helps avoid other unnecessary meetings
- Three questions answered by each team member:
  1. What did you do yesterday?
  2. What will you do today?
  3. What obstacles are in your way?

# 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].

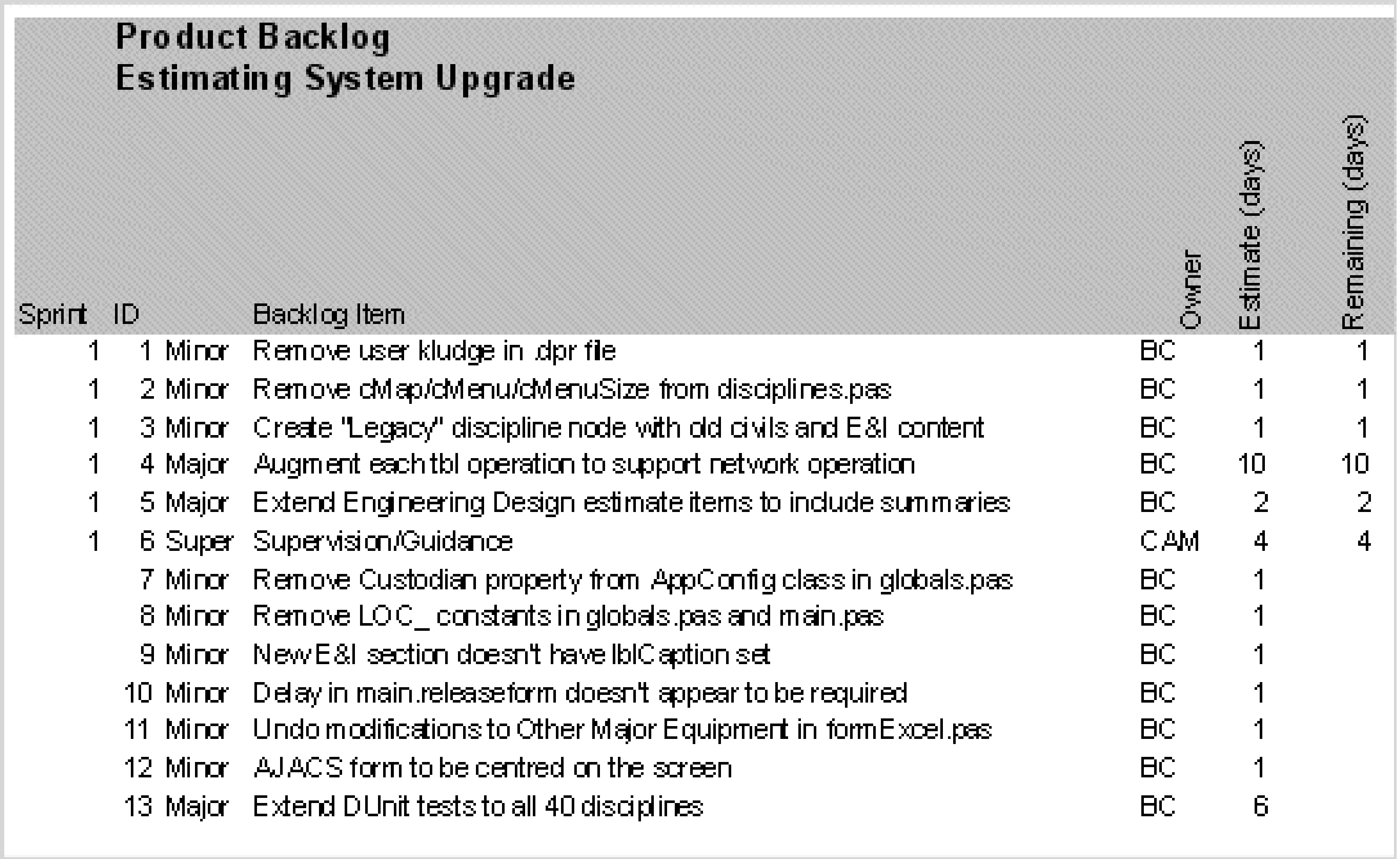
- Example:
  - "As a user, I want to log in, so I can access subscriber content."

- **story points**: Rating of effort needed to implement this story
  - common scales: 1-10, shirt sizes (XS, S, M, L, XL), etc.

# Sample Product Backlog

| Backlog item | Estimate |
|---|---|
| Allow a guest to make a reservation | 3 (story points) |
| As a guest, I want to cancel a reservation. | 5 |
| As a guest, I want to change the dates of a reservation. | 3 |
| As a hotel employee, I can run RevPAR reports (revenue-per-available-room) | 8 |
| Improve exception handling | 8 |
| ... | 30 |
| ... | 50 |

# Sample Product Backlog 2



**Product Backlog**
**Estimating System Upgrade**

| Sprint | ID | | Backlog Item | Owner | Estimate (days) | Remaining (days) |
|---|---|---|---|---|---|---|
| 1 | 1 | Minor | Remove user kludge in .dpr file | BC | 1 | 1 |
| 1 | 2 | Minor | Remove cMap/cMenu/cMenuSize from disciplines.pas | BC | 1 | 1 |
| 1 | 3 | Minor | Create "Legacy" discipline node with old civils and E&I content | BC | 1 | 1 |
| 1 | 4 | Major | Augment each tbl operation to support network operation | BC | 10 | 10 |
| 1 | 5 | Major | Extend Engineering Design estimate items to include summaries | BC | 2 | 2 |
| 1 | 6 | Super | Supervision/Guidance | CAM | 4 | 4 |
| | 7 | Minor | Remove Custodian property from AppConfig class in globals.pas | BC | 1 | |
| | 8 | Minor | Remove LOC_ constants in globals.pas and main.pas | BC | 1 | |
| | 9 | Minor | New E&I section doesn't have lblCaption set | BC | 1 | |
| | 10 | Minor | Delay in main.releaseform doesn't appear to be required | BC | 1 | |
| | 11 | Minor | Undo modifications to Other Major Equipment in formExcel.pas | BC | 1 | |
| | 12 | Minor | AJACS form to be centred on the screen | BC | 1 | |
| | 13 | Major | Extend DUnit tests to all 40 disciplines | BC | 6 | |

# Sprint Backlog

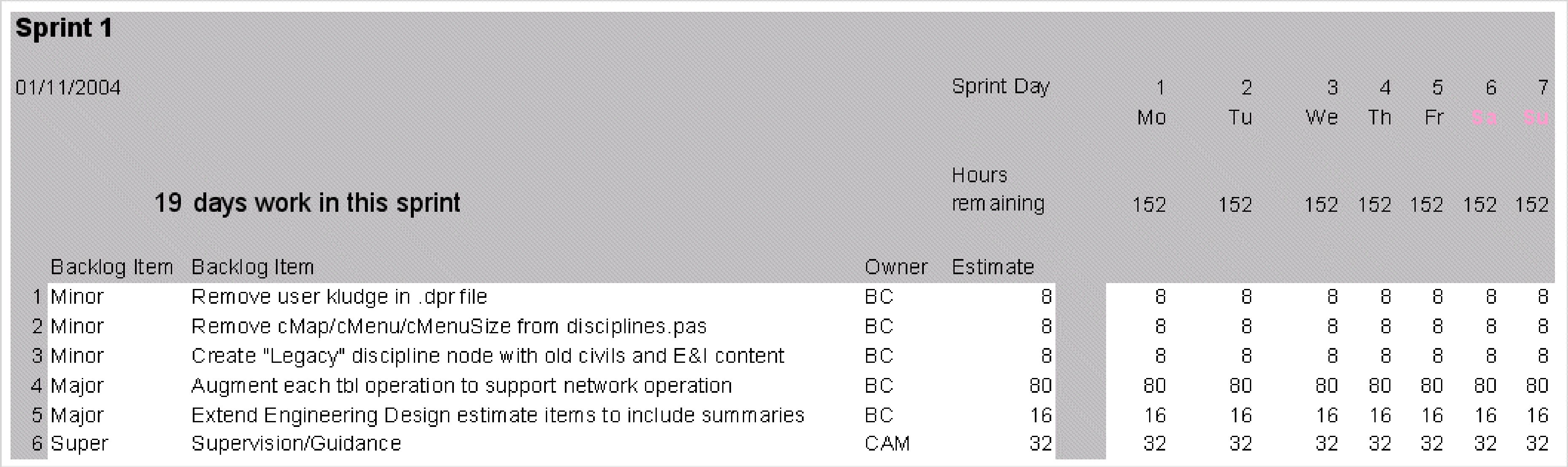
- Individuals sign up for work of their own choosing
  - Work is never assigned
- Estimated work remaining is updated daily

- 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

# Sample Sprint backlog

| Tasks | Mon | Tue | Wed | Thu | Fri |
|---|---|---|---|---|---|
| Code the user interface | 8 | 4 | 8 | | |
| Code the middle tier | 16 | 12 | 10 | 4 | |
| Test the middle tier | 8 | 16 | 16 | 11 | 8 |
| Write online help | 12 | | | | |
| Write the Foo class | 8 | 8 | 8 | 8 | 8 |
| Add error logging | | | 8 | 4 | |

# Sample Sprint Backlog

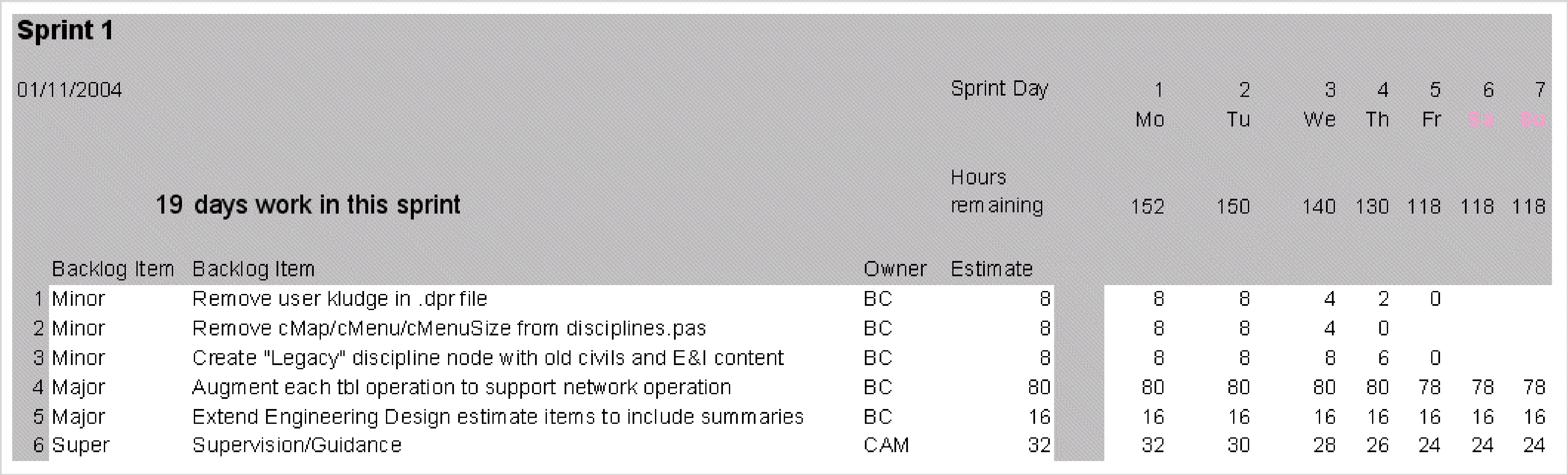


**Sprint 1**

01/11/2004

**19 days work in this sprint**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Sprint Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | | | | | Mo | Tu | We | Th | Fr | Sa | Su |
| | | | | Hours remaining | 152 | 152 | 152 | 152 | 152 | 152 | 152 |
| | Backlog Item | Backlog Item | Owner | Estimate | | | | | | | |
| 1 | Minor | Remove user kludge in .dpr file | BC | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| 2 | Minor | Remove cMap/cMenu/cMenuSize from disciplines.pas | BC | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| 3 | Minor | Create "Legacy" discipline node with old civils and E&I content | BC | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 |
| 4 | Major | Augment each tbl operation to support network operation | BC | 80 | 80 | 80 | 80 | 80 | 80 | 80 | 80 |
| 5 | Major | Extend Engineering Design estimate items to include summaries | BC | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 6 | Super | Supervision/Guidance | CAM | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 |



**Sprint 1**

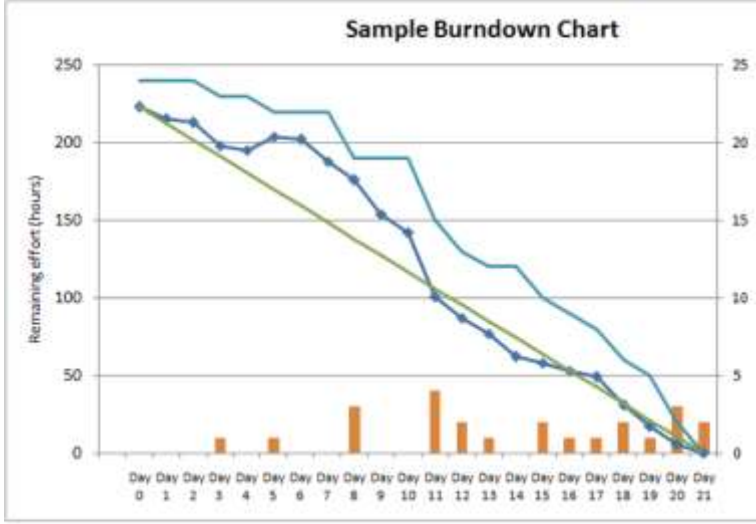
01/11/2004

**19 days work in this sprint**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Sprint Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | | | | | Mo | Tu | We | Th | Fr | Sa | Su |
| | | | | Hours remaining | 152 | 150 | 140 | 130 | 118 | 118 | 118 |
| | Backlog Item | Backlog Item | Owner | Estimate | | | | | | | |
| 1 | Minor | Remove user kludge in .dpr file | BC | 8 | 8 | 8 | 4 | 2 | 0 | | |
| 2 | Minor | Remove cMap/cMenu/cMenuSize from disciplines.pas | BC | 8 | 8 | 8 | 4 | 0 | | | |
| 3 | Minor | Create "Legacy" discipline node with old civils and E&I content | BC | 8 | 8 | 8 | 8 | 6 | 0 | | |
| 4 | Major | Augment each tbl operation to support network operation | BC | 80 | 80 | 80 | 80 | 80 | 78 | 78 | 78 |
| 5 | Major | Extend Engineering Design estimate items to include summaries | BC | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 16 |
| 6 | Super | Supervision/Guidance | CAM | 32 | 32 | 30 | 28 | 26 | 24 | 24 | 24 |

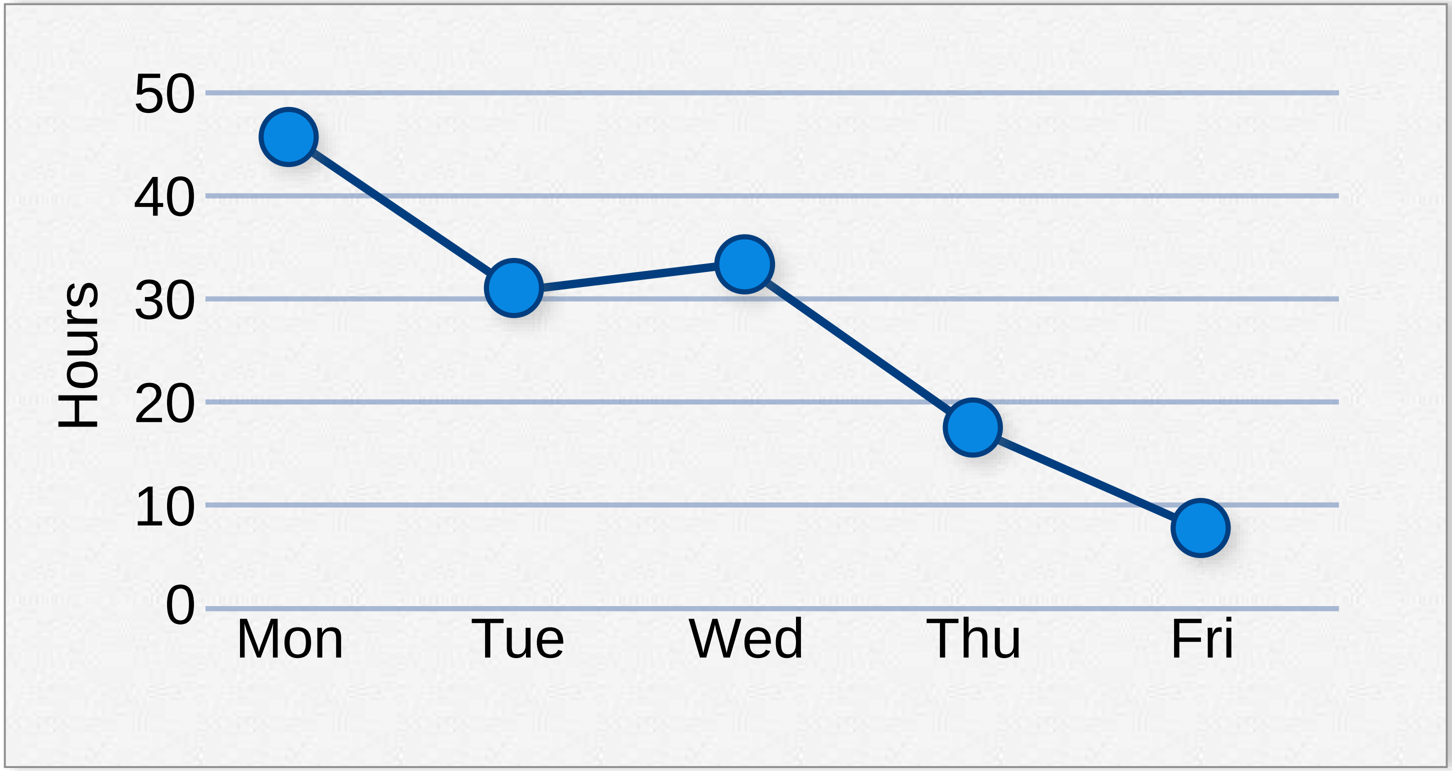
# Sprint Burndown Chart

- 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)



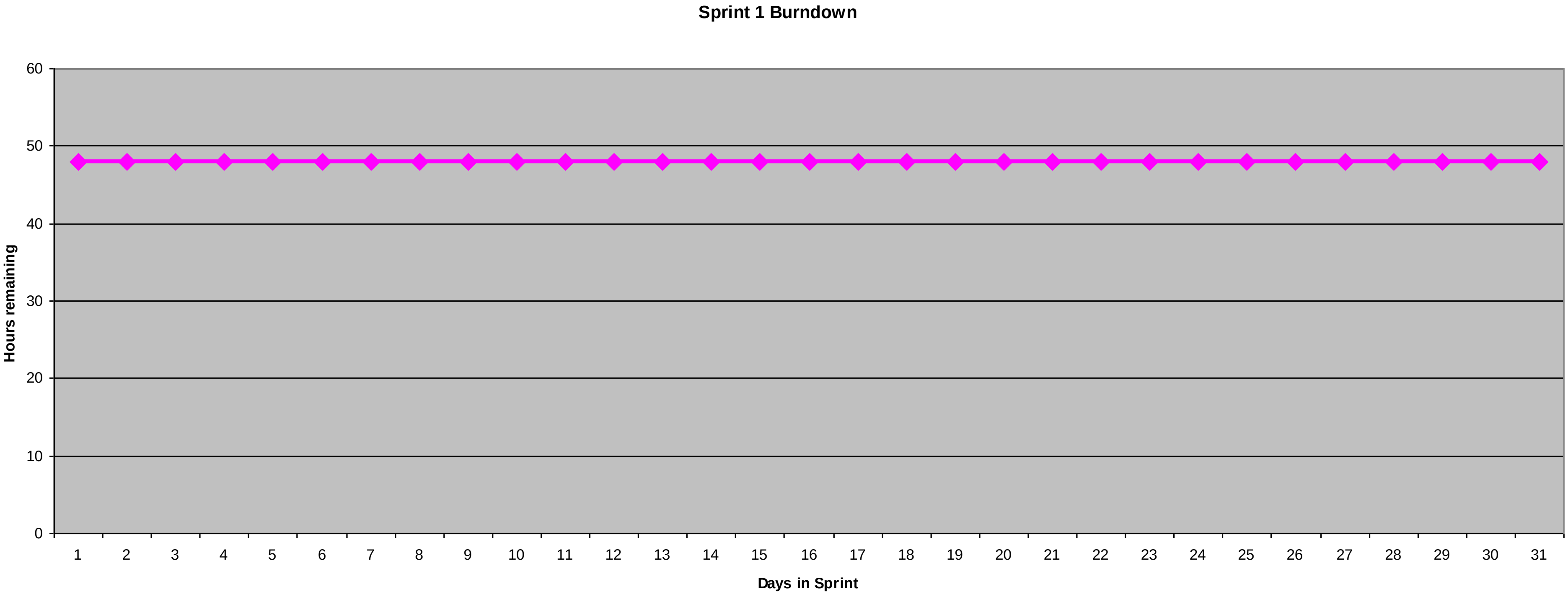
- *variation:* Release burndown chart
  - shows overall progress
  - updated at end of each sprint

| 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 | | | | |



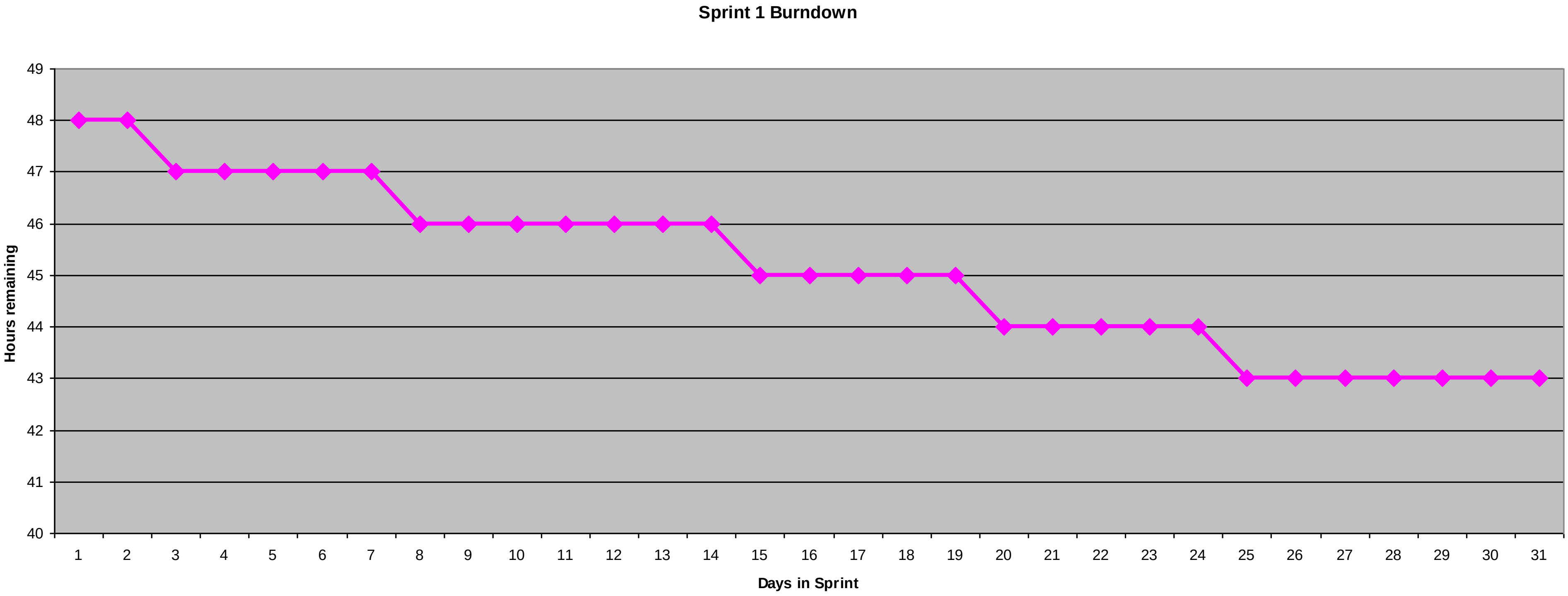
# Burndown Example 1

No work being performed



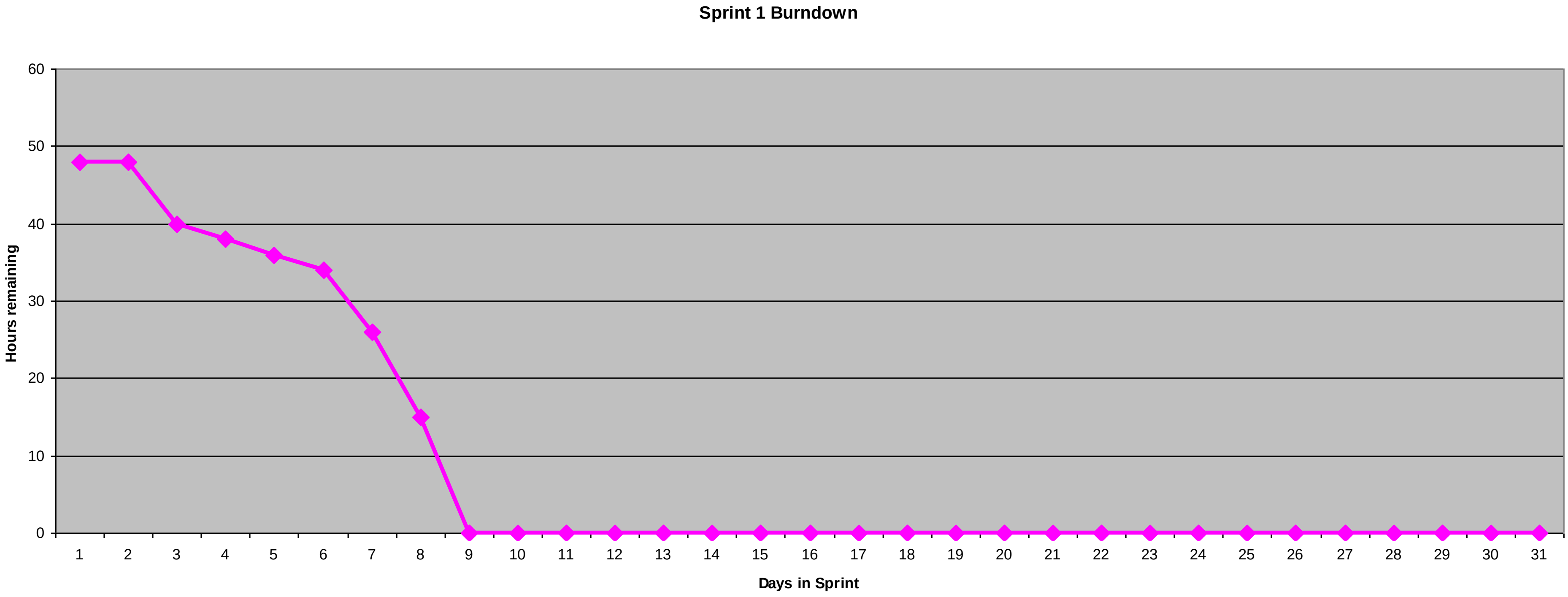
# Burndown Example 2

Work being performed, but not fast enough



# Burndown Example 3

Work being performed, but too fast!



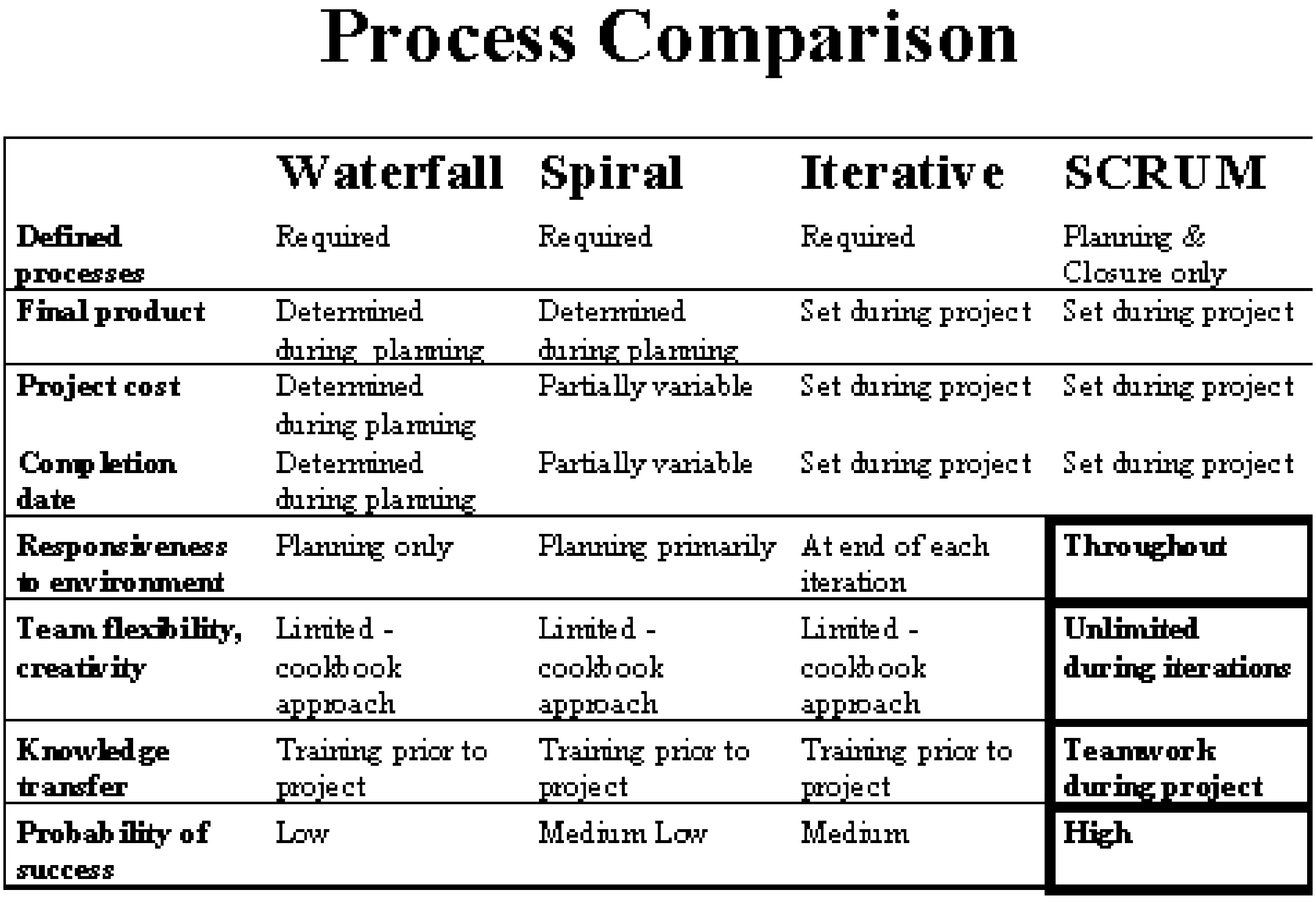
# The Sprint Review

- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features or underlying architecture
- Informal
  - 2-hour prep time rule
  - No slides
- Whole team participates
- Invite the world

# Scalability

- Typical individual team is 7 ± 2 people
  - Scalability comes from teams of teams
- Factors in scaling
  - Type of application
  - Team size
  - Team dispersion
  - Project duration
- Scrum has been used on multiple 500+ person projects

# Scrum vs. Other Models



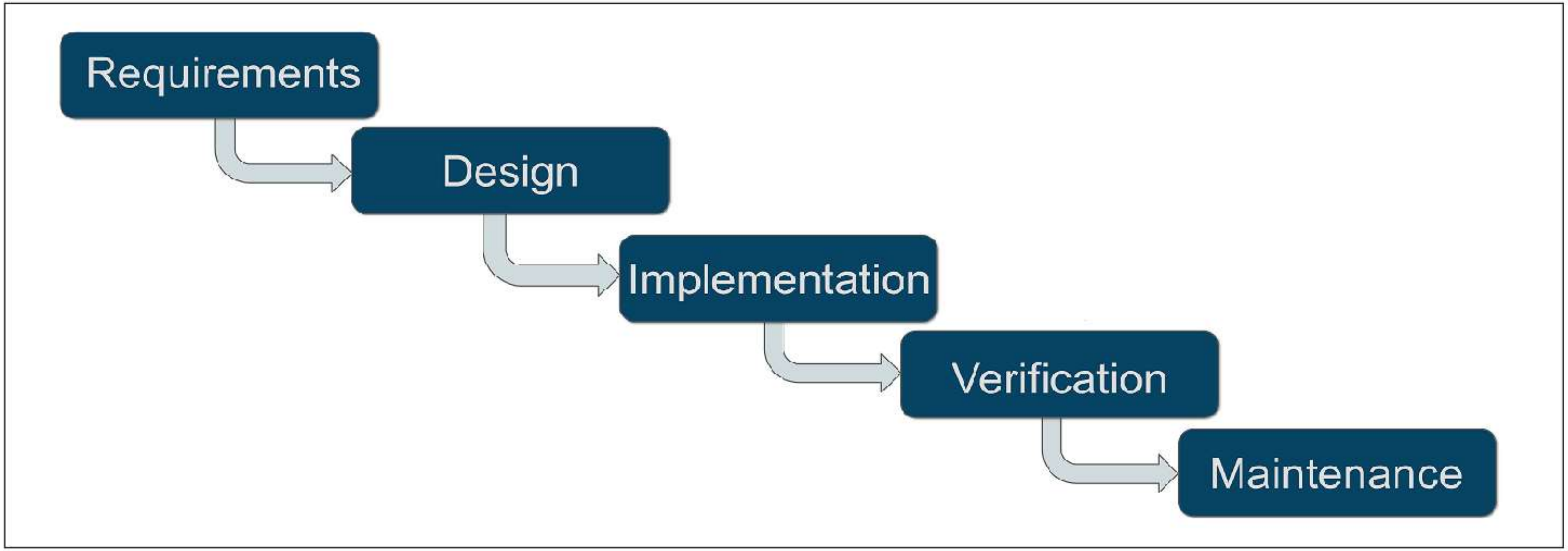
## Process Comparison

| | Waterfall | Spiral | Iterative | SCRUM |
|---|---|---|---|---|
| **Defined processes** | Required | Required | Required | Planning & Closure only |
| **Final product** | Determined during planning | Determined during planning | Set during project | Set during project |
| **Project cost** | Determined during planning | Partially variable | Set during project | Set during project |
| **Completion date** | Determined during planning | Partially variable | Set during project | Set during project |
| **Responsiveness to environment** | Planning only | Planning primarily | At end of each iteration | **Throughout** |
| **Team flexibility, creativity** | Limited - cookbook approach | Limited - cookbook approach | Limited - cookbook approach | **Unlimited during iterations** |
| **Knowledge transfer** | Training prior to project | Training prior to project | Training prior to project | **Teamwork during project** |
| **Probability of success** | Low | Medium Low | Medium | **High** |

# DevOps

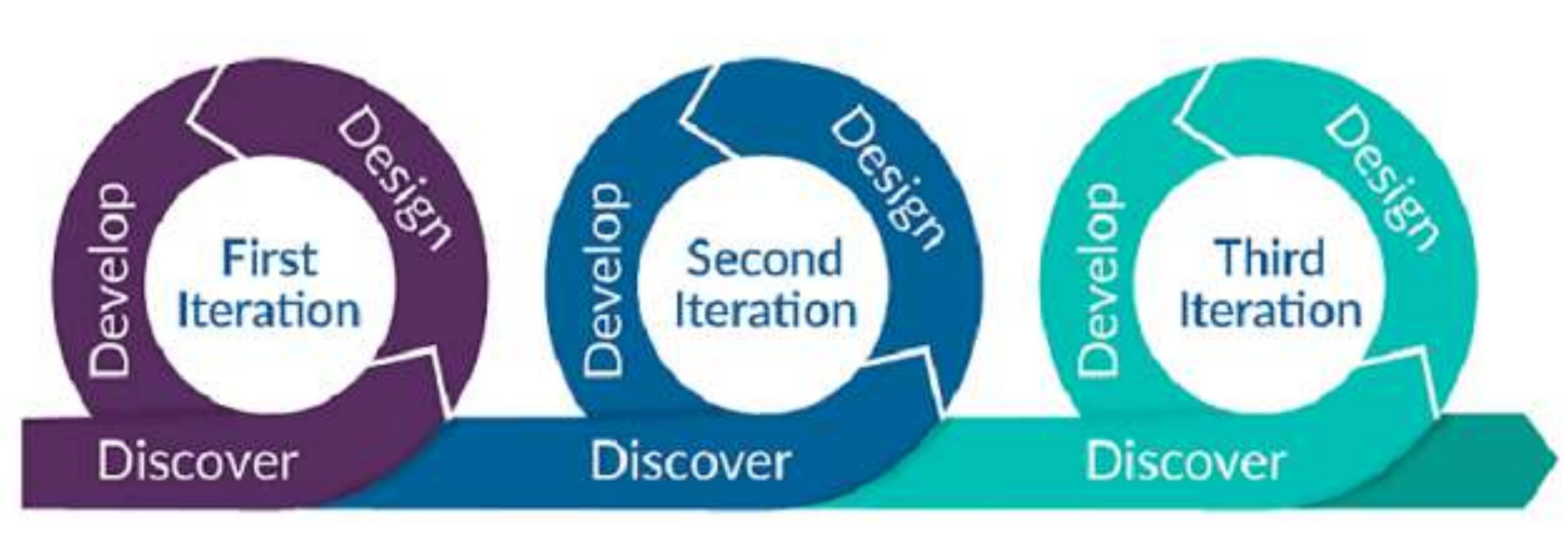
Development and Operations

# Waterfall Model



Source: https://www.edureka.co/blog/devops-lifecycle/

# Agile Model

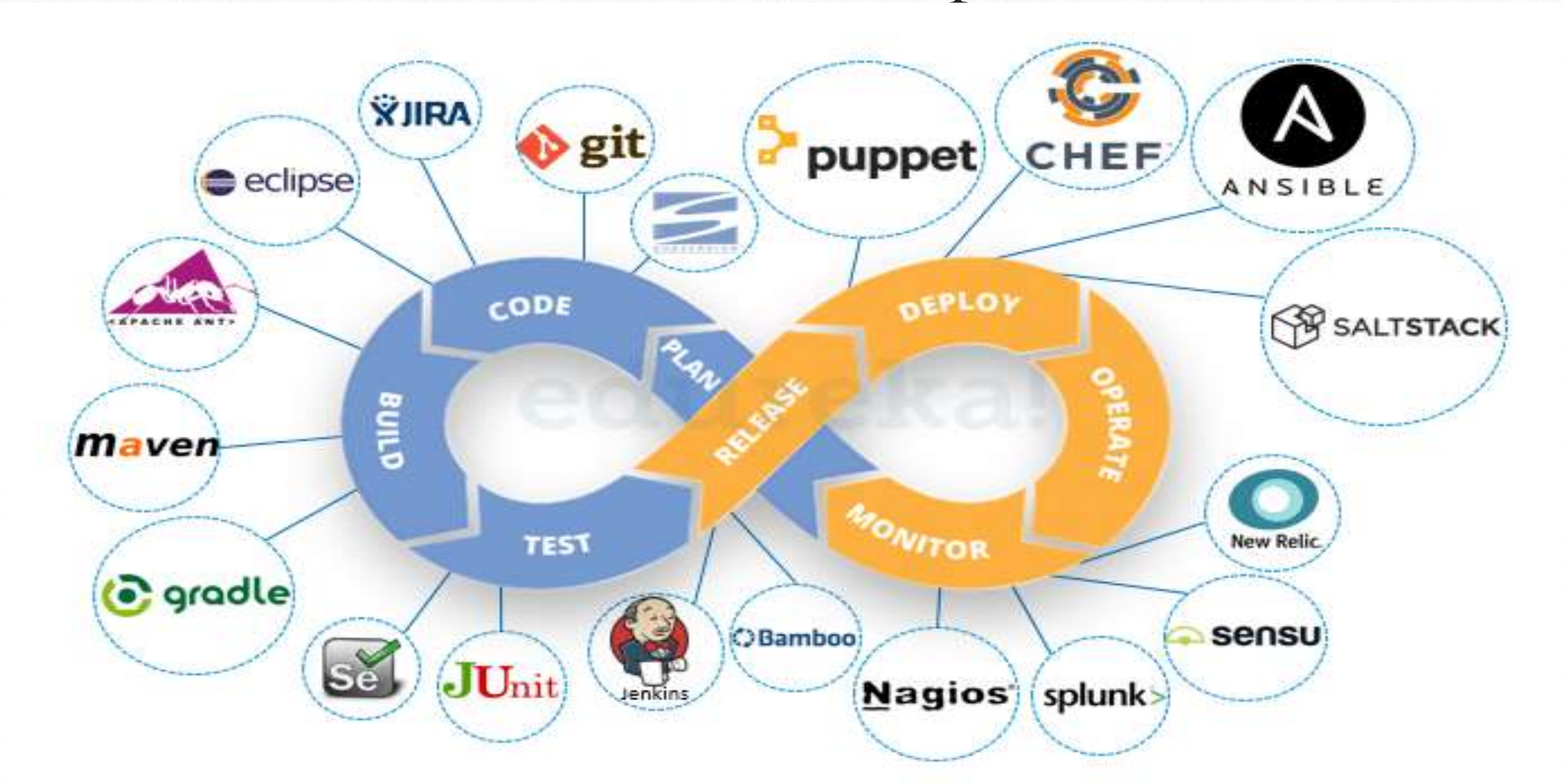


Source: https://www.edureka.co/blog/devops-lifecycle/

# Why DevOps

- Lack of collaboration between Developers and Operation Engineers
- Slowed down the development process and releases
- Better collaboration between the teams and faster delivery of software
- DevOps enabled continuous software delivery with less complex problems to fix and faster resolution of problems

# What is DevOps?

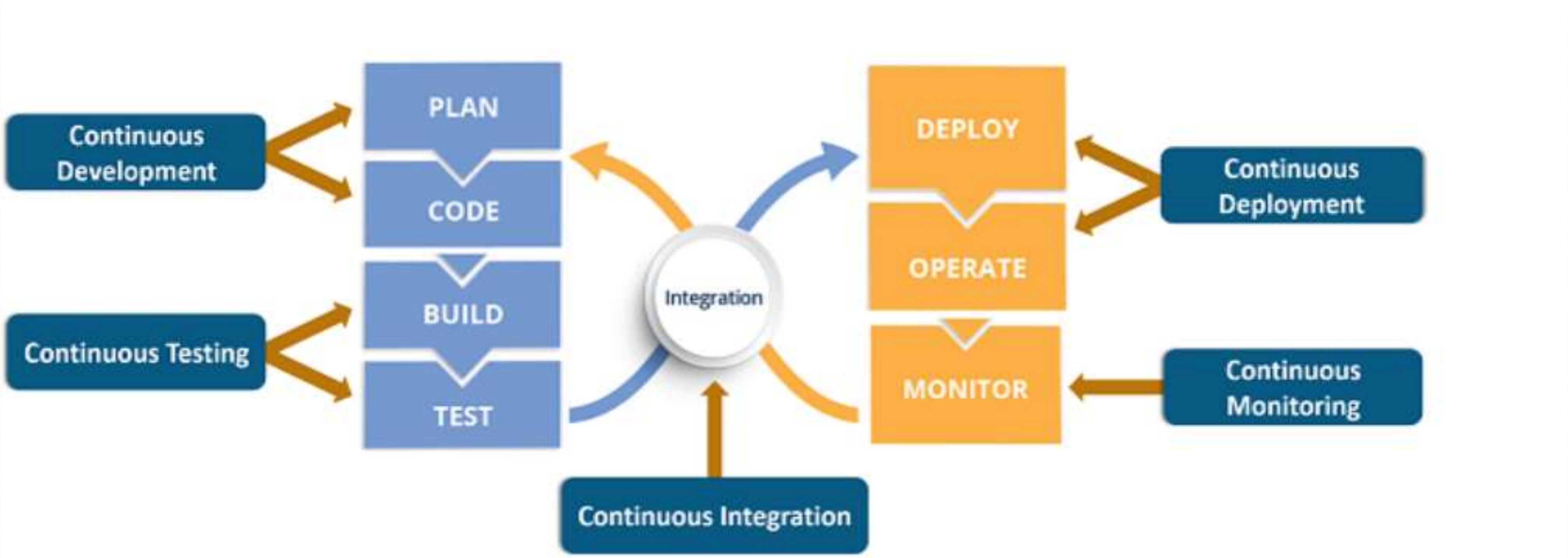


Source: https://www.edureka.co/blog/devops-lifecycle/

# What is DevOps?

- Practice that allows a single team to manage the entire application development life cycle
- To shorten the system’s development life cycle while delivering features, fixes, and updates frequently in close alignment with business objectives.
- Approach through which superior quality software can be developed quickly and with more reliability.
- Stages such as continuous development, continuous integration, continuous testing, continuous deployment, and continuous monitoring

# What is DevOps Life cycle?



Source: https://www.edureka.co/blog/devops-lifecycle/

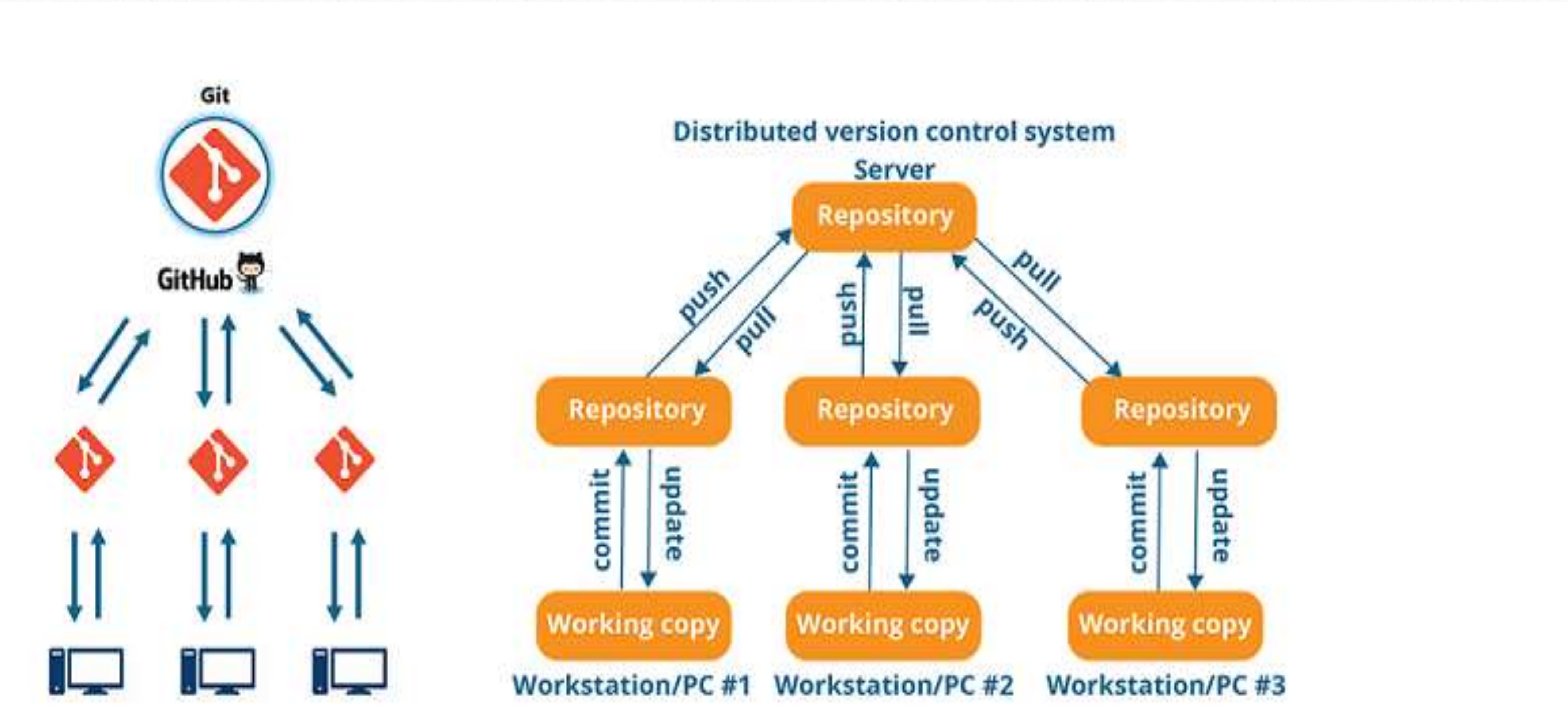
# Continuous Development

- The phase that involves ‘planning‘ and ‘coding‘ of the software
- The vision of the project is decided during the planning phase and the developers begin developing the code for the application.
- A number of tools for maintaining the code
- Code is maintained by using Version Control tools
- Maintaining the code is referred to s Source Code Management.
- Git, SVN, Mercurial, CVS, and JIRA. Also tools like Ant, Maven, gradle can be used in this phase for building/ packaging the code into an executable file

# Continuous Development

- Git is a distributed version control tool
- Supports distributed non-linear workflows by providing data assurance for developing quality software
- Tools like Git enable communication between the development and the operations team
- Developing a large project with a huge number of collaborators, - communication is important while making changes in the project
- Commit messages and a stable version of the code
- Hence, Git plays a vital role in succeeding at DevOps

# Continuous Development

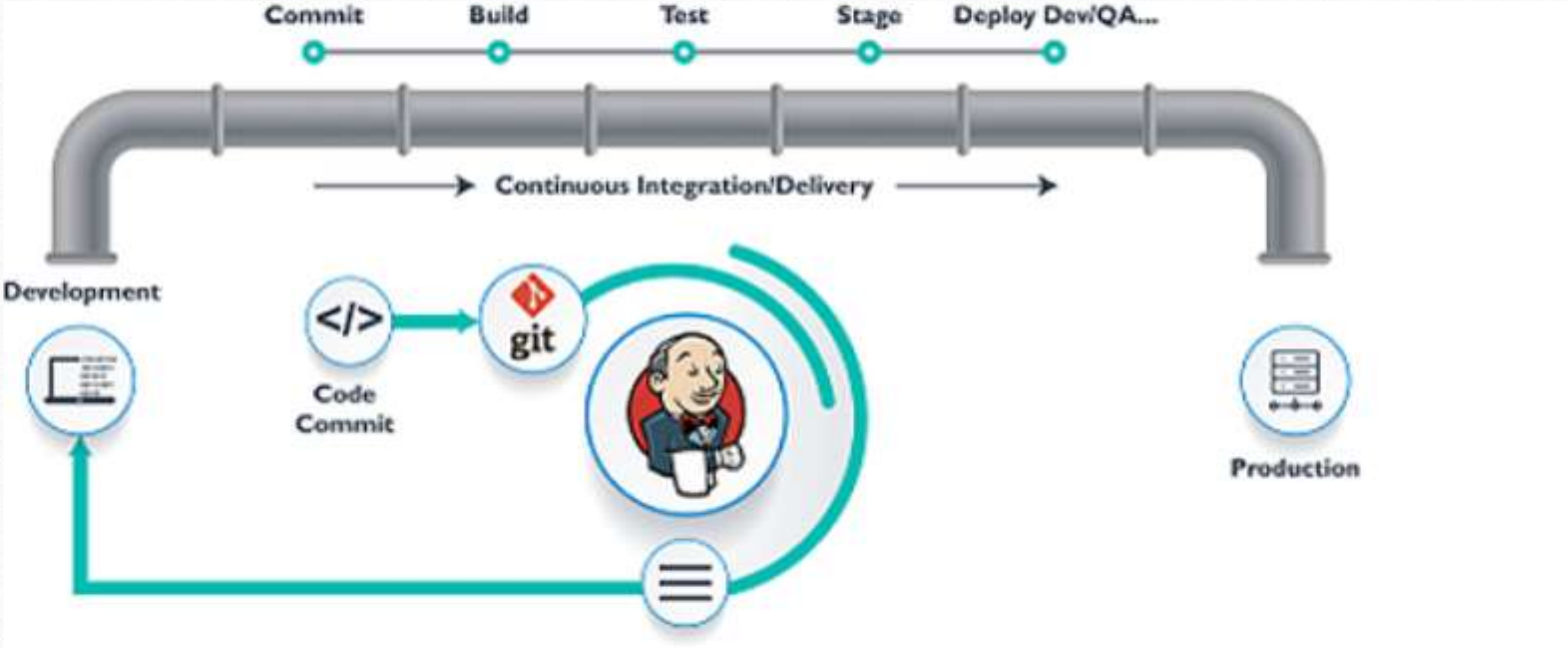


# Continuous Testing

- This is the stage where the developed software is continuously tested for bugs.
- Automation testing  tools like Selenium, TestNG, JUnit, etc are used.  (test multiple code-bases thoroughly in parallel)

- Docker Containers can be used for simulating the test DevOps
- Automation testing saves a lot of time, effort and labor for executing the tests instead of doing this manually.
- Report generation is a big plus. The task of evaluating the test cases that failed in a test suite gets simpler.
- We can also schedule the execution of the test cases at predefined times. After testing, the code is continuously integrated with the existing code.

# Continuous Integration



# Continuous Deployment

- Configuration Management CM is the act of establishing and maintaining consistency in an application’s functional requirements and performance

- CM -execute tasks quickly and frequently. Some popular tools that are used here are Puppet, Chef, SaltStack, and Ansible

- tools help produce consistency across Development, Test, Staging and Production environments

# Continuous Monitoring

- Monitor the performance of application
- The system errors such as low memory, server not reachable, etc are resolved in this phase
- It maintains the security and availability of the services & network issues.
- The popular tools used for this are Splunk, ELK Stack, Nagios, NewRelic and Sensu
- Improve productivity and increase the reliability of the systems