# MODULE – I Contents

- Introduction of Software Engineering
- Software Process
  - WATERFALL MODEL
  - V-MODEL
  - INCREMENTAL MODEL
  - PROTOTYPE MODEL
  - SPIRAL MODEL
  - AGILE MODEL (Extreme Programming)

# Software Definition

Software is:

(1) instructions (computer programs) that when executed provide desired features, function, and performance.

(2) data structures that enable the programs to adequately manipulate information.

(3) documentation that describes the operation and use of the programs.

# Software Engineering

- The **software** is a collection of integrated programs.
- Computer programs and related documentation such as requirements, design models and user manuals.
- **Engineering** is the application of scientific and practical knowledge to invent, design, build, maintain, and improve frameworks, processes, etc.
- **Software Engineering** is an engineering branch related to the evolution of software product using well defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

# Software Engineering

- Some realities:
  - a concerted effort should be made to understand the problem before a software solution is developed
  - design becomes a pivotal activity
  - software should exhibit high quality
  - software should be maintainable
- The seminal definition:
  - [Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

# Software Engineering

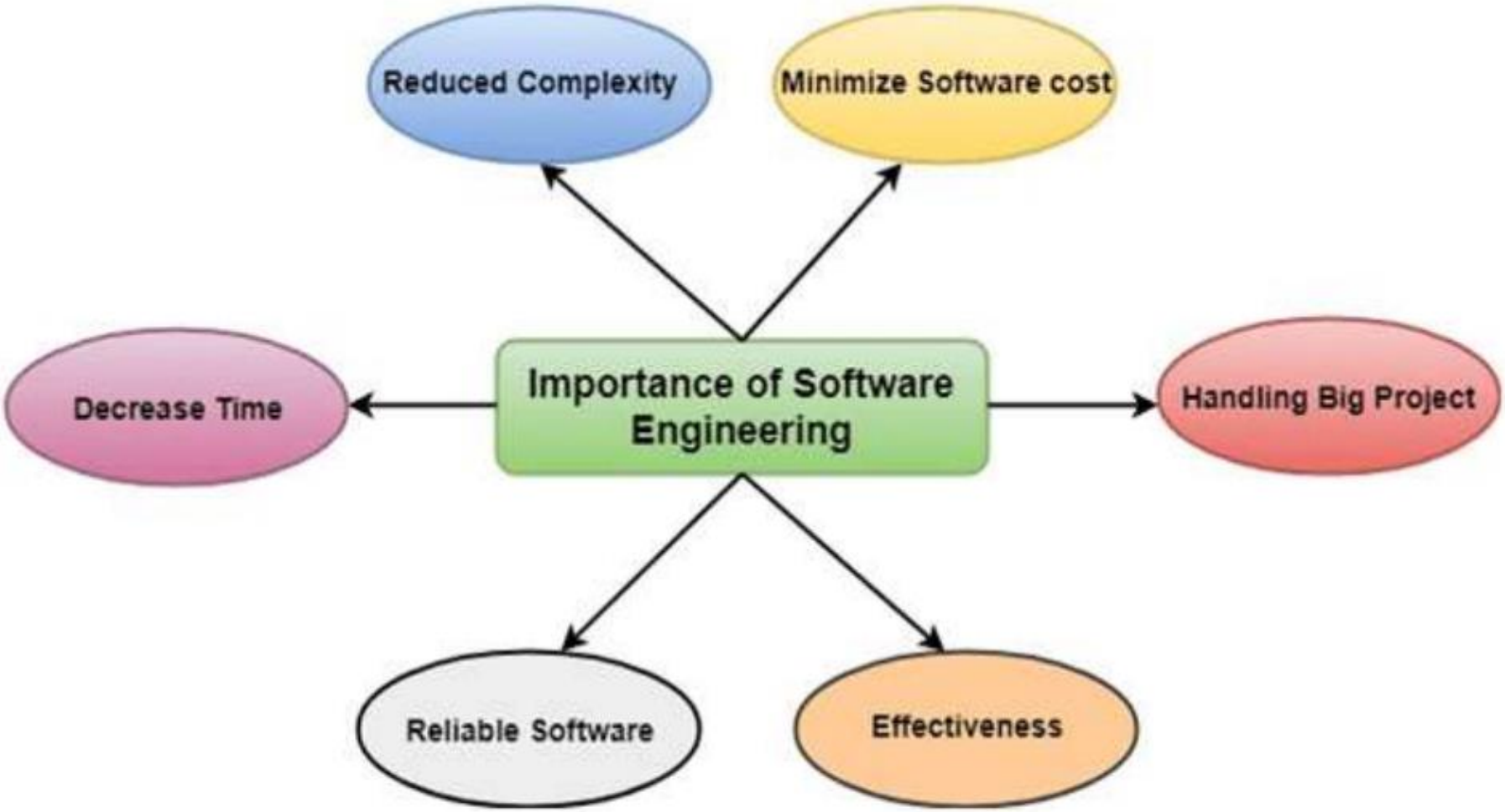
The IEEE definition of Software Engineering:

- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

# Need of Software Engineering

- Huge Programming
- Adaptability
- Cost
- Dynamic Nature
- Quality Management

# Importance of Software Engineering



# Characteristics of Software Engineer

- Exposure to systematic methods, i.e., familiarity with **software engineering principles**.
- Good **technical knowledge** of the project range (Domain knowledge).
- Good **programming abilities**.
- Good **communication skills**. These skills comprise of oral, written, and interpersonal skills.
- High **motivation**.
- Sound knowledge of fundamentals of **computer science**.
- Intelligence.
- Ability to **work in a team** Discipline, etc.

# Characteristics of Software Process

1. **Functionality:** Software systems should function correctly, i.e. perform all the functions for which they are designed .
2. **Reliability:** ability to perform its intended functions correctly and consistently over time.
3. **Efficiency:** ability to use resources such as memory, processing power, and network bandwidth in an optimal way.
4. **Usability:** the amount of effort or time required to learn how to use the software.
5. **Maintainability:** Software should be easy to repair, improve and correct errors.
6. **Portability:** the ability to use software in different environments
7. **Integrity:** the software cannot be modified without authorization
8. **Flexibility** : ability of the software solution to adapt to potential or future changes in its requirements.

# Software Applications

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- Webapps (web applications)
- AI software

# Software Process

- A process is a **collection of activities** that are performed when some work product is to be created.
- A process framework establishes the **foundation** for a complete software engineering process by **identifying a small number of framework activities** that are applicable to all software projects, regardless of their size or complexity.
- Software engineering process framework activities are complemented by several **umbrella activities**.
- In general, umbrella activities are applied throughout a software project and help a software team **manage and control progress, quality, change, and risk**.

# Software Process

**Framework Activities:**

1. Communication
2. Planning
3. Modeling
   i. Analysis
   ii. Design
4. Construction
   i. Code generation
   ii. Testing
5. Deployment

**Phases of Software Development Life Cycle (SDLC):**

1. Requirement Gathering
2. Planning
3. Analysis
4. Design
5. Implementation
6. Testing
7. Maintenance

# Software Process- Umbrella Activities

- Software project tracking and control
- Risk management
- Software quality assurance
- Measurement
- Technical Reviews
- Software configuration management
- Reusability management
- Work product preparation and production

# PROCESS MODELS

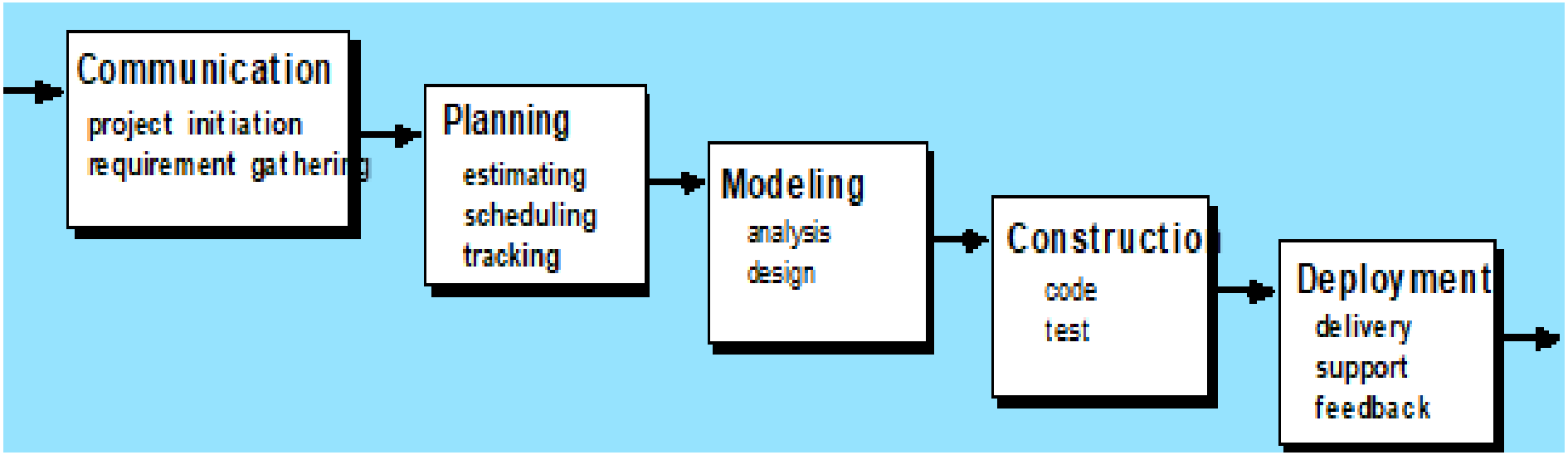
- A process model provides a specific roadmap for software engineering work.
- It defines the **flow of all activities**, actions and tasks, the degree of iteration, the work products, and the organization of the work that must be done.

# Process Models Types

1. The Waterfall Model.
2. V-Model
3. Incremental Model
4. Prototyping Model
5. Spiral Model
6. AGILE MODEL

# 1.The Waterfall Model

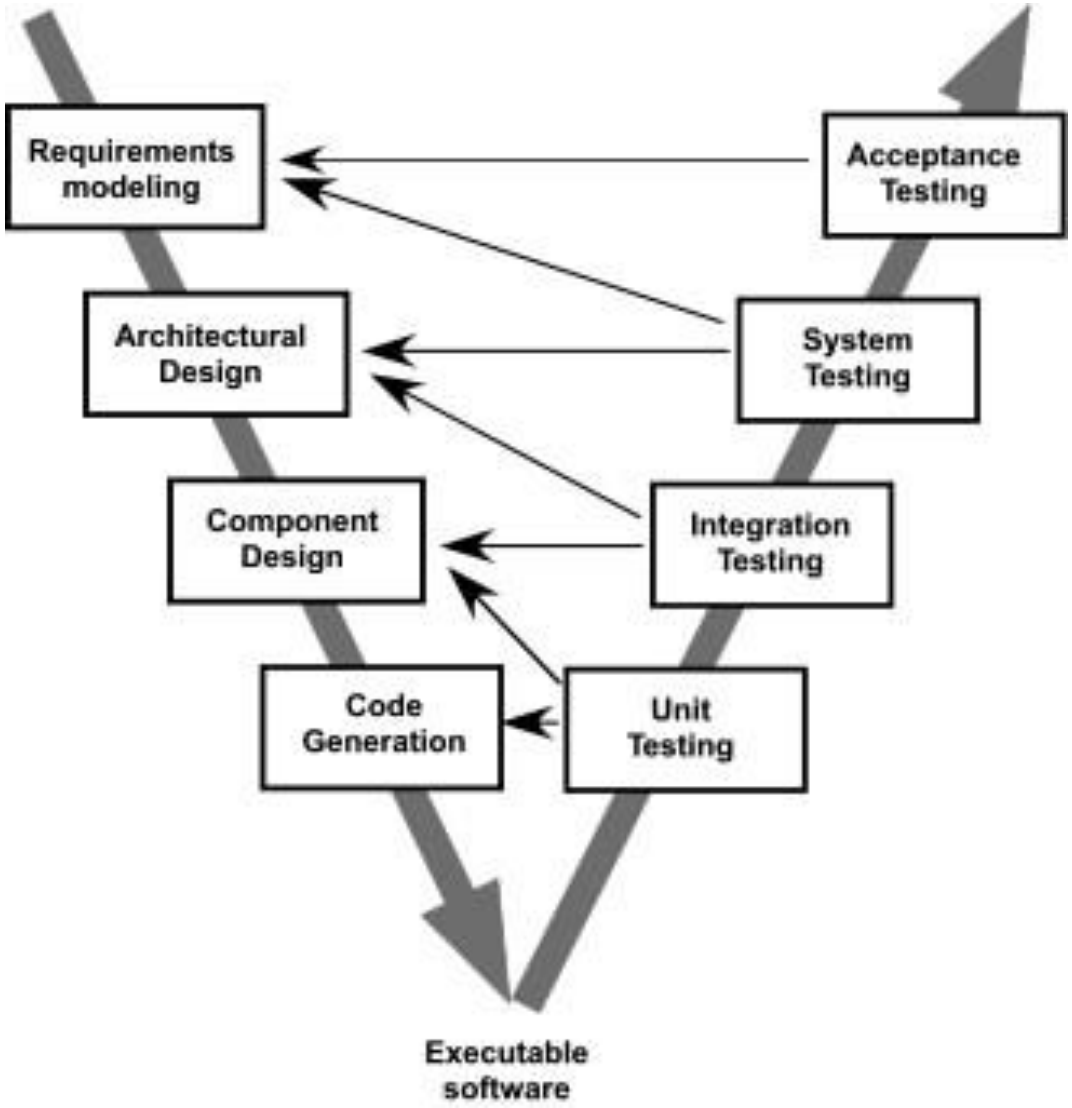
- The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development.
- **A useful process model in situations where requirements are fixed, and work is to proceed to completion in a linear manner.**



# Problems encountered by Waterfall model

- Real projects rarely follow the **sequential flow** that the model proposes
- It is often **difficult** for the customer to state **all requirements** explicitly
- The customer must have **patience** because working version of the program(s) will not be available until late in the project time span.
- Classic life cycle leads to “blocking states” in which some project team members must **wait** for other members of the team to complete dependent tasks.

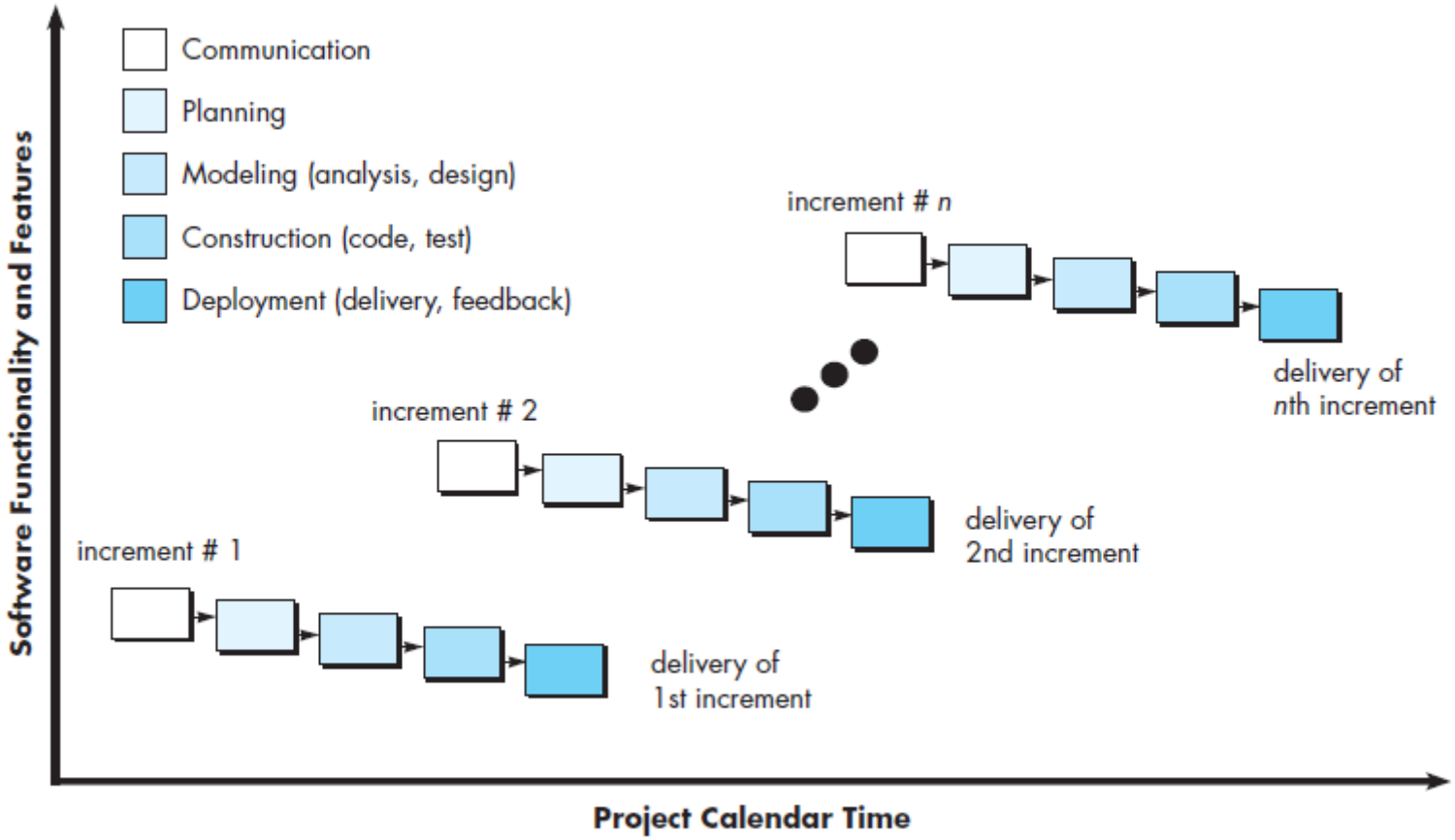
# 2. The V-Model




V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities

The V-model illustrates how verification and validation actions are associated with earlier engineering actions.

# 3. The Incremental Model



# The Incremental Model

- There may be a compelling need to provide a limited set of software functionality to users quickly and then refine and expand on that functionality in later software releases
- The incremental model combines elements of linear and parallel process flows
- When an incremental model is used, the first increment is often a core product.
- The incremental process model focuses on the delivery of an operational product with each increment.

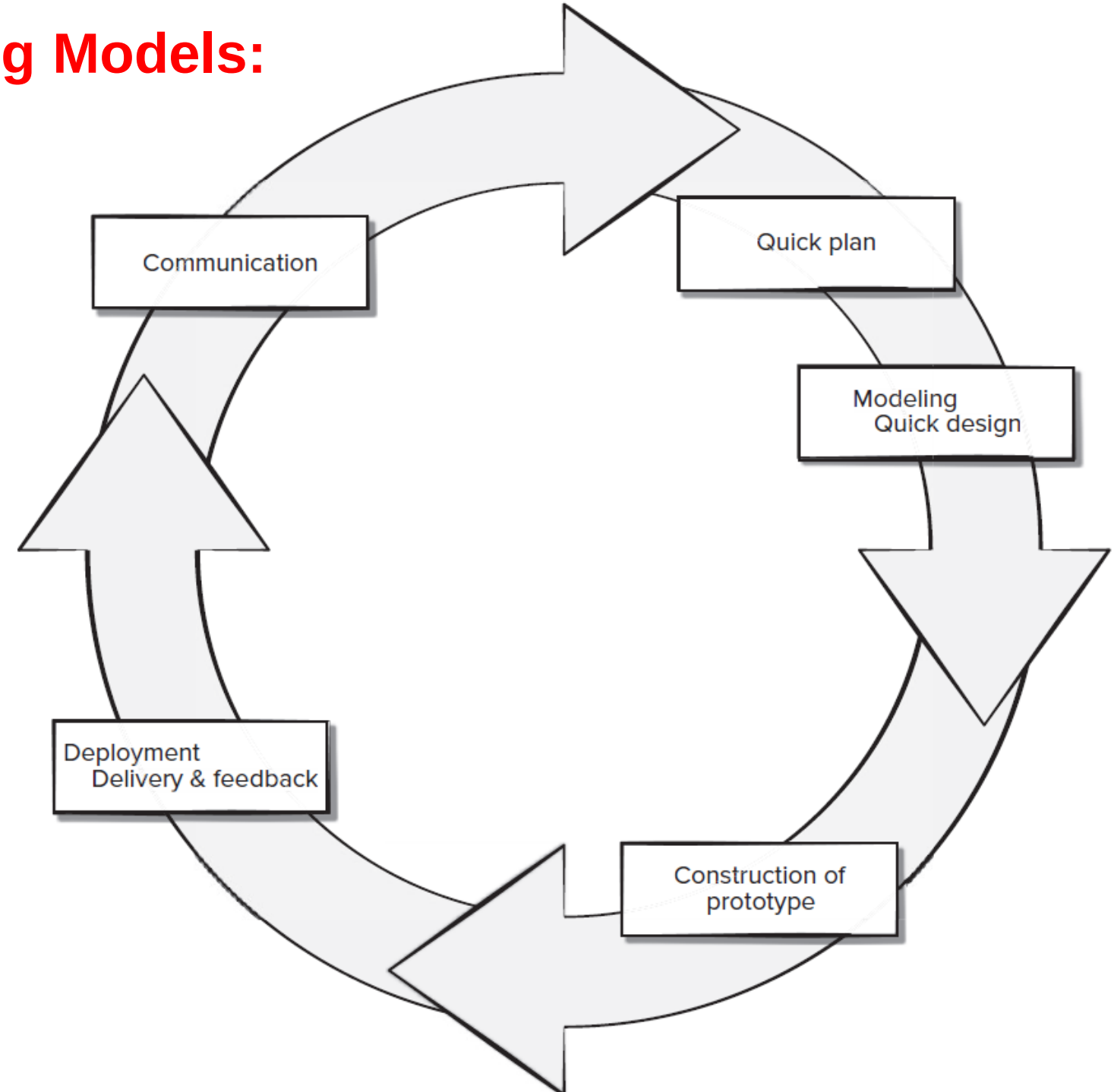
# 4. Prototyping Models

- Prototyping paradigm can offer the best approach If developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take
- Prototyping paradigm assists you and other stakeholders to better understand what is to be built when requirements are fuzzy

Construction of prototype

- Stakeholders should agree that the prototype is built to serve as a mechanism for defining requirements

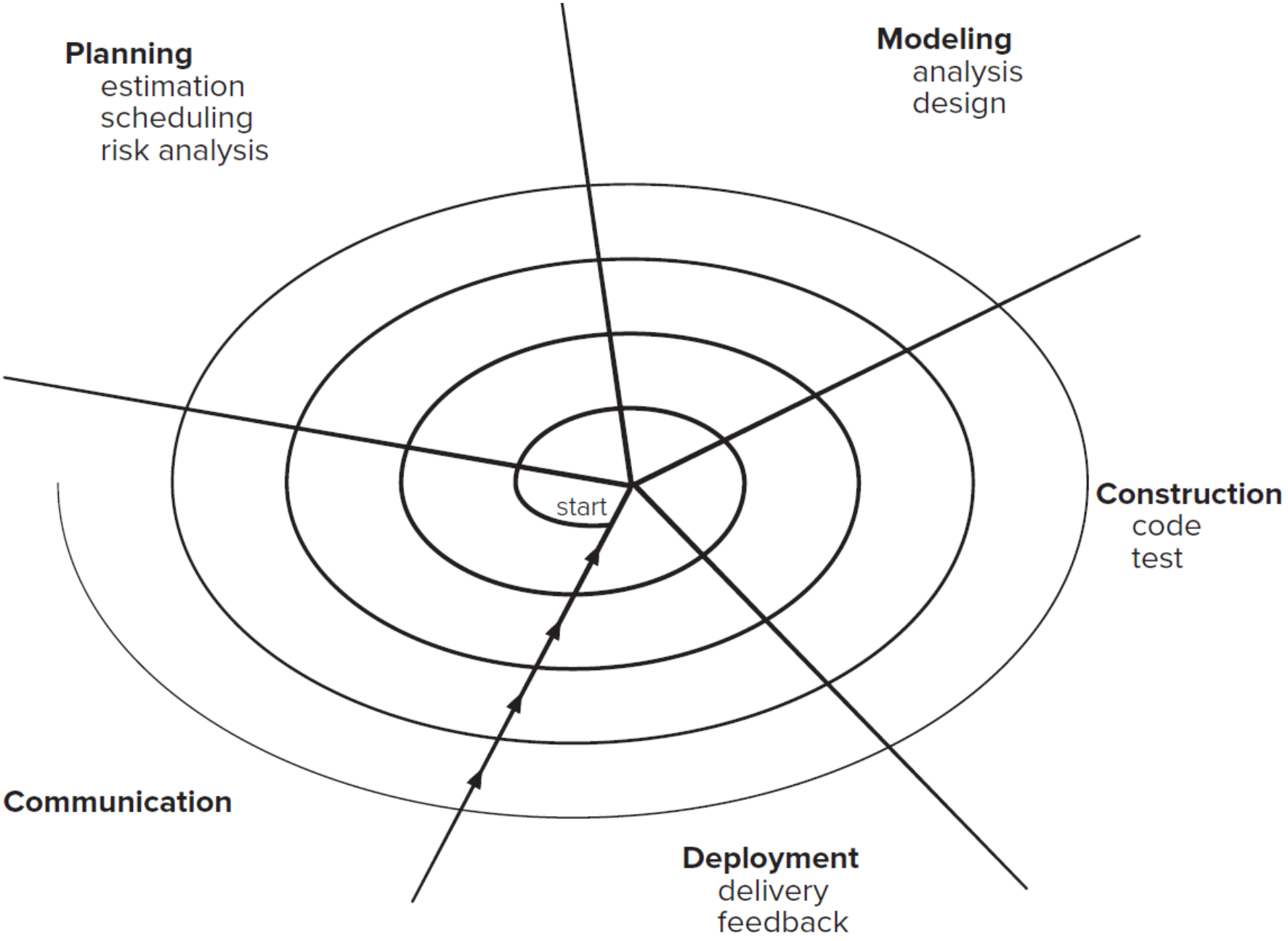
# Prototyping Models:



# 5.The Spiral Model

- Proposed by Barry Boehm in 1988. Is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model
- Spiral model is divided into a set of framework activities defined by the software engineering team
- Each of the framework activities represent one segment of the spiral path
- Anchor point milestones—a combination of work products and conditions that are attained along the path of the spiral
- Product specification -> Prototype -> Sophisticated version of software.

# The Spiral


Planning
estimation
scheduling
risk analysis
Modeling
analysis
design
start
Construction
code
test
Communication
Deployment
delivery
feedback

# 6. Agile software development

- Agile software engineering combines a philosophy and a set of development guidelines.

- The philosophy encourages customer satisfaction and early incremental delivery of software; small, highly motivated project teams; informal methods; minimal software engineering work products; and overall development simplicity.

- The development guidelines stress delivery over analysis and design (although these activities are not discouraged), and active and continuous communication between developers and customers.

# Why is it important?

- The modern business environment that spawn's computer-based systems and software products is fast-paced and ever-changing.
- Agile software engineering represents a reasonable alternative to conventional software engineering for certain classes of software and certain types of software projects.
- It has been demonstrated to deliver successful systems quickly.

# What is “Agility”?

- Agility has become today’s buzzword when describing a modern software process.
- Everyone is agile.
- An agile team is a nimble team able to appropriately respond to changes.
- Change is what software development is very much about.
- Changes in the software being built,
- changes to the team members,
- changes because of new technology and
- changes of all kinds that may have an impact on the product they build or the project that creates the product.

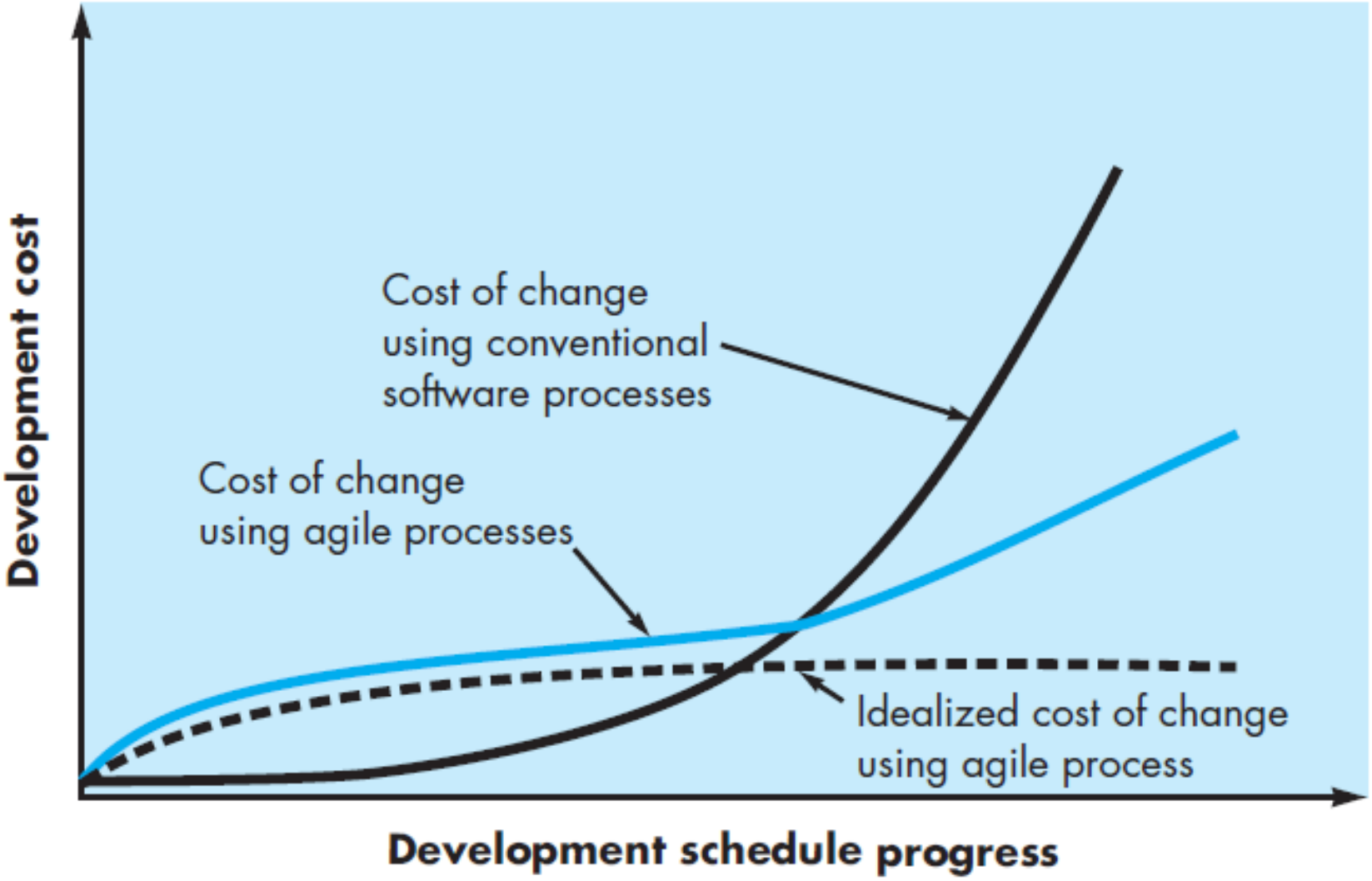
# What is “Agility”?

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Yielding …

- Rapid, incremental delivery of software

# Agility and the Cost of Change



# An Agile Process

- Is 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 multiple ‘software increments’
- Adapts as changes occur

## What are the steps?

- Agile development might best be termed “software engineering lite.” The basic framework activities—communication, planning, modeling, construction, and deployment—remain.
- But they morph into a minimal task set that pushes the project team toward construction and delivery

# Agility Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Businesspeople and developers must work together daily throughout the project.

# Agility Principles…

5. Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

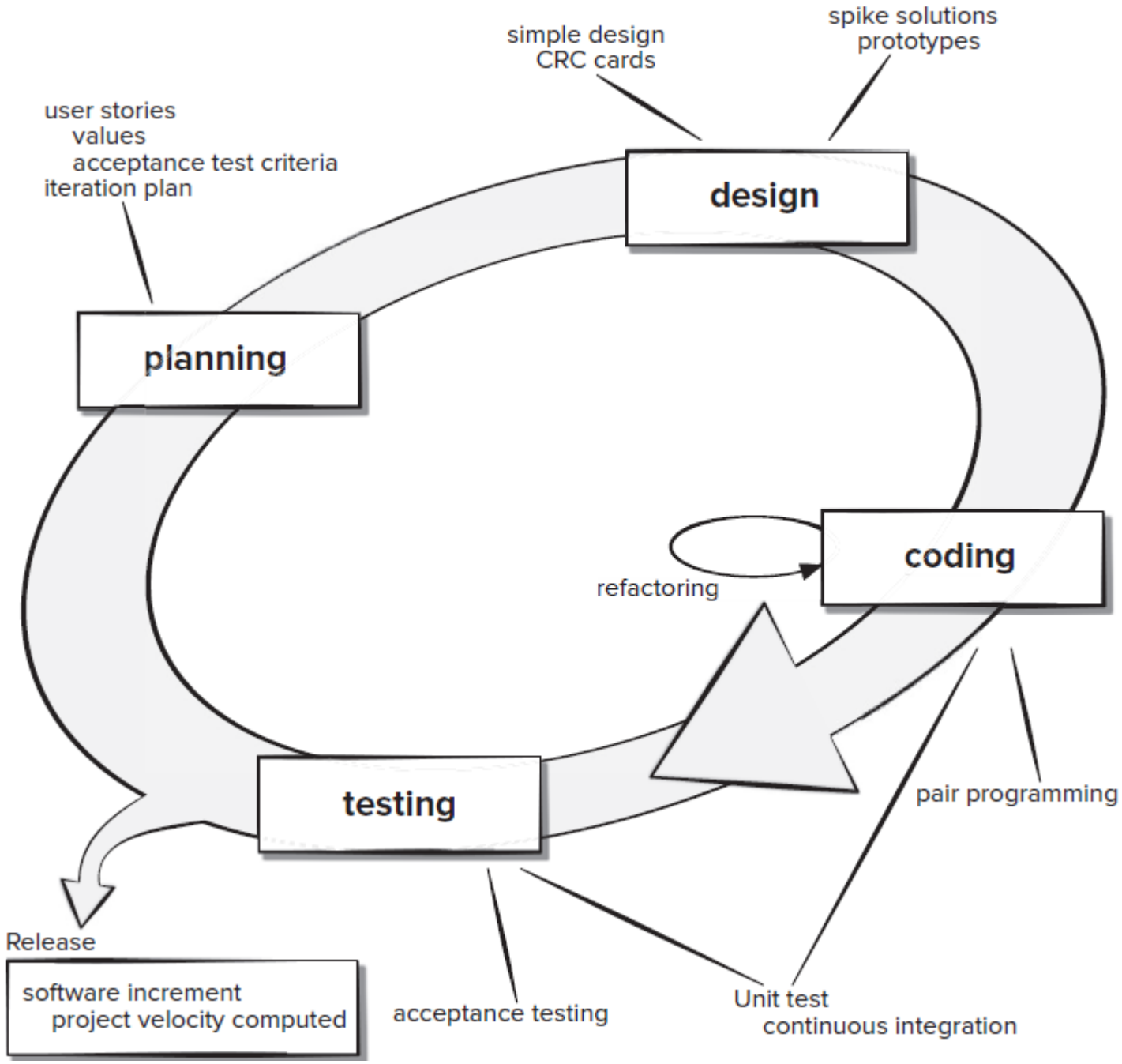
# Agility Principles…

9. **Continuous attention** to technical excellence and good design enhances agility.
10. **Simplicity** the art of maximizing the amount of work not done – is essential.
11. The **best architectures, requirements, and designs** emerge from self–organizing teams.
12. At regular intervals, the **team** reflects on how to become **more effective**, then tunes and adjusts its behavior accordingly.

# Extreme Programming (XP)

- Extreme Programming (XP), the most widely used approach to agile software development, originally proposed by Kent Beck
- Extreme Programming uses an object-oriented approach as its preferred development paradigm and encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding, and testing.

# Extreme Programming (XP)



# Extreme Programming (XP)

- XP Planning
  - Begins with the creation of “user stories” Listening leads to the creation of a set of “ stories ” that describe required output, features, and functionality for software to be built.
  - Agile team assesses each story and assigns a cost
  - Stories are grouped to for a deliverable increment
  - A commitment is made on delivery date
  - After the first increment “project velocity” is used to help define subsequent delivery dates for other increments

# Extreme Programming (XP)

- XP Design
  - Follows the KIS principle (keep it simple)
  - Encourage the use of CRC cards (*Class-responsibility-collaborator)*
  - For difficult design problems, suggests the creation of "spike solutions"—a design prototype (*simplest possible program to explore potential solutions*)
  - Encourages "refactoring"—an iterative refinement of the internal program design (*systematic process of improving existing computer code, without adding new functionality or changing external behavior of the code*)
- XP Coding
  - Recommends the construction of a unit test for a store *before* coding commences
  - Encourages "pair programming"
- XP Testing
  - All unit tests are executed daily and "Acceptance tests" are defined by the customer and executed to assess customer visible functionality

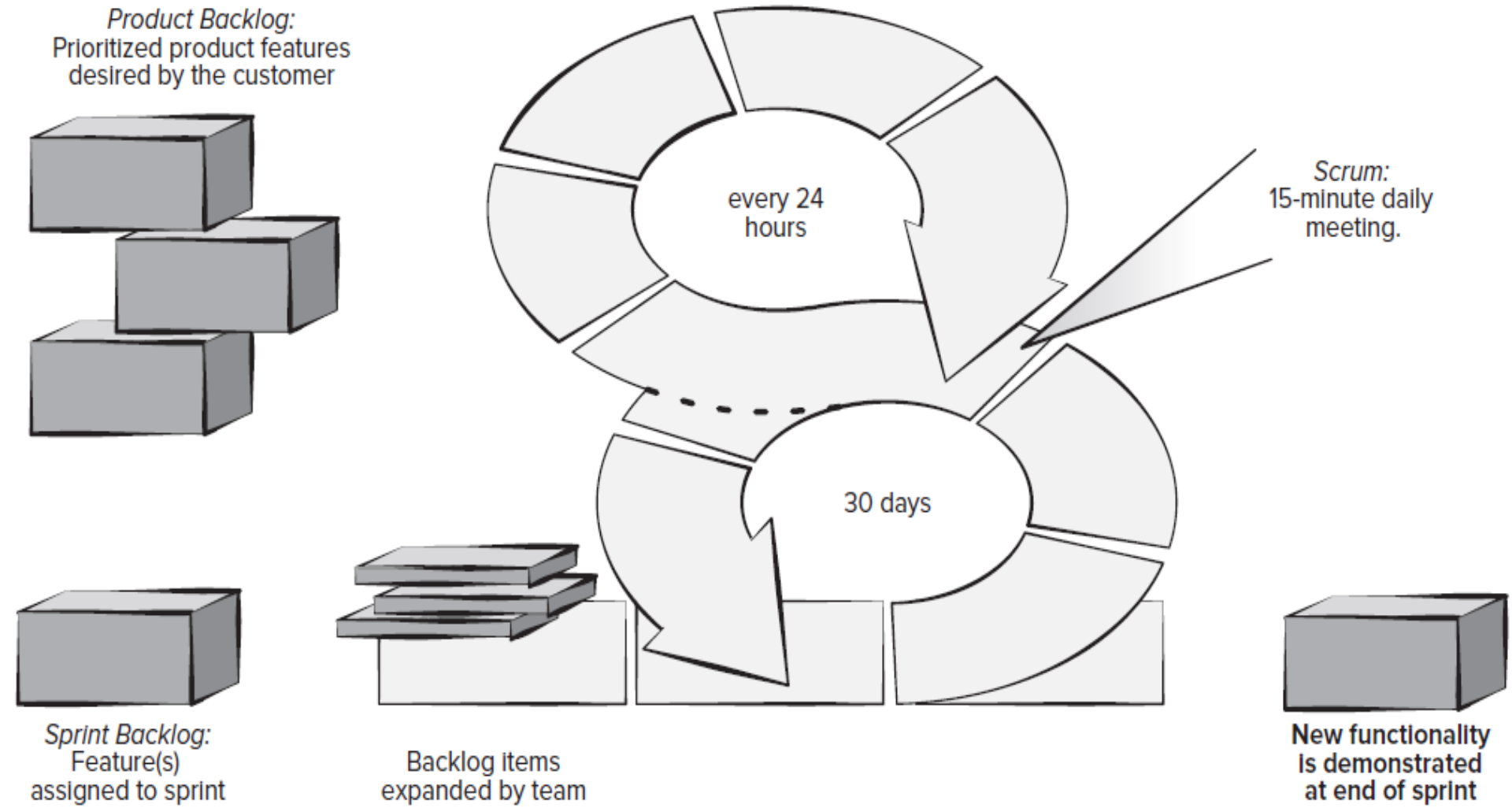
Good practices need to be practiced in extreme programming: Some of the good practices that have been recognized in the extreme programming model and suggested to maximize their use are given below.

- Code Review: Code review detects and corrects errors efficiently. It suggests pair programming
- Testing: Testing code helps to remove errors and improves its reliability. XP suggests test-driven development (TDD) to continually write and execute test cases.
- Incremental development: Incremental development is very good because customer feedback is gained and based on this development team produces 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.
- Design: Good quality design is important to develop good quality software.
- Integration testing: Extreme programming suggests that the developers should achieve continuous integration by building and performing integration testing several times a day.

# Scrum

- A software development method Originally proposed by Schwaber and Beedle (an activity occurs during a rugby match) in early 1990.
- Scrum—distinguishing features
  - Development work is partitioned into “packets”
  - Testing and documentation are on-going as the product is constructed
  - Work units occurs in “sprints” and is derived from a “backlog” of existing changing prioritized requirements
  - Changes are not introduced in sprints (short term but stable) but in backlog.
  - Meetings are very short (15 minutes daily) and sometimes conducted without chairs ( what did you do since last meeting? What obstacles are you encountering? What do you plan to accomplish by next meeting?)
  - “demos” are delivered to the customer with the time-box allocated. May not contain all functionalities. So customers can evaluate and give feedbacks. a

# Scrum


*Product Backlog:*
Prioritized product features
desired by the customer
every 24
hours
*Scrum:*
15-minute daily
meeting.
30 days
*Sprint Backlog:*
Feature(s)
assigned to sprint
Backlog items
expanded by team
New functionality
is demonstrated
at end of sprint