

# Fundamental of Software Engineering CSE 3205

## **Chapter Two** **Software processes**

*School of Electrical Engineering and  
Computing*

*Department of computer Science and  
Engineering*

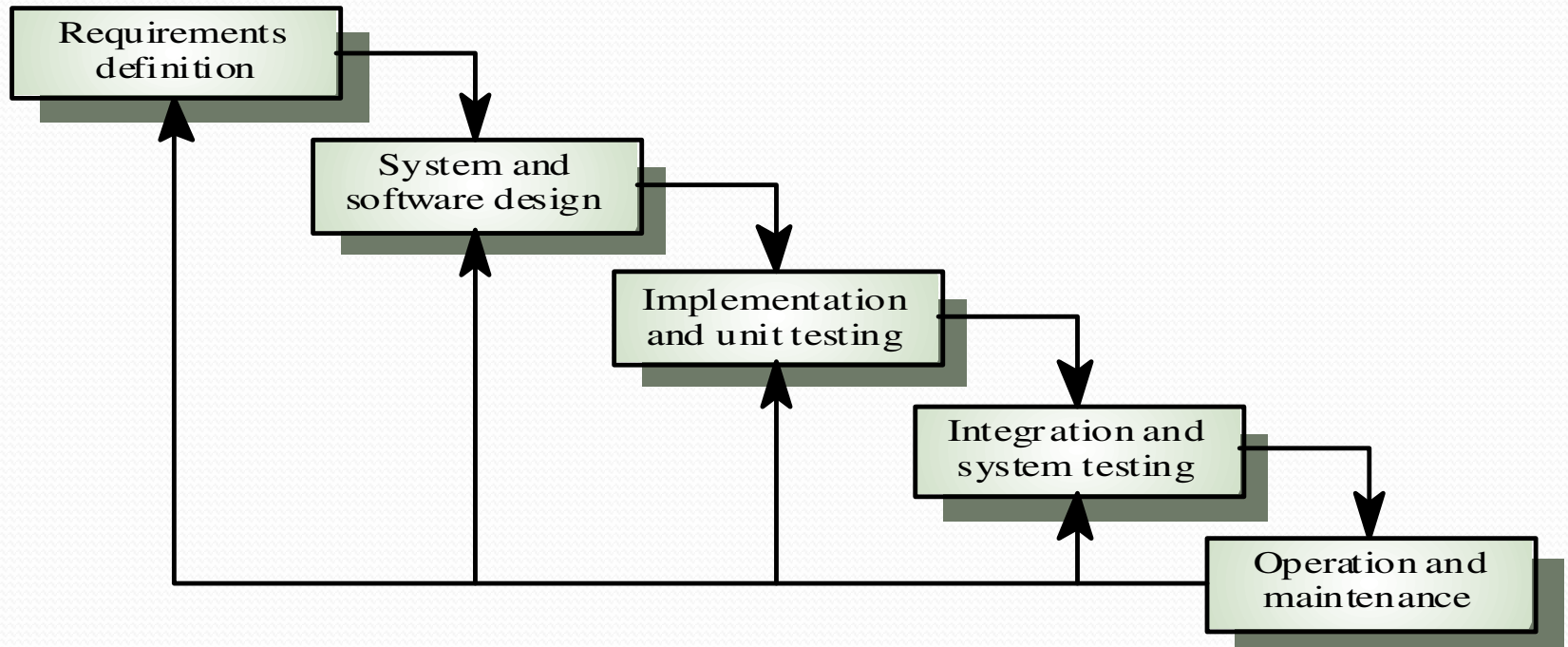
*ASTU*

# The Software Process

- **A structured set of activities required to develop a software system**
  - Requirement Identification
  - Specification
  - Analysis, design and implementation.
  - Validation
  - Evolution
- **A software process model is an abstract representation of a process**
  - It presents a description of a process from some particular perspective

# Process Model

## 1. Waterfall Model



# Waterfall model phases

- **Requirements** analysis and definition
- System and software **design**
- **Implementation** and unit testing
- **Integration** and system testing
- **Operation** and maintenance

The drawback of the waterfall model is the difficulty of accommodating change after the process is underway

# Waterfall model Requirement and Design

Artefacts produced in the requirements and Design phases

- **SRS** - **S**oftware **R**equirements **s**pecification document
- SRS might include:
  - User usage stories (scenarios) – **Use cases**.
  - Static analysis – **class diagrams**.
  - Behavioural analysis – **sequence diagrams, state charts**.

The specification and design activities are heavily time consuming.

# Waterfall model problems

- **Inflexible partitioning** of the project into distinct stages
  - Difficult to respond to changing customer requirements
- 

This model is only appropriate when the requirements are well-understood

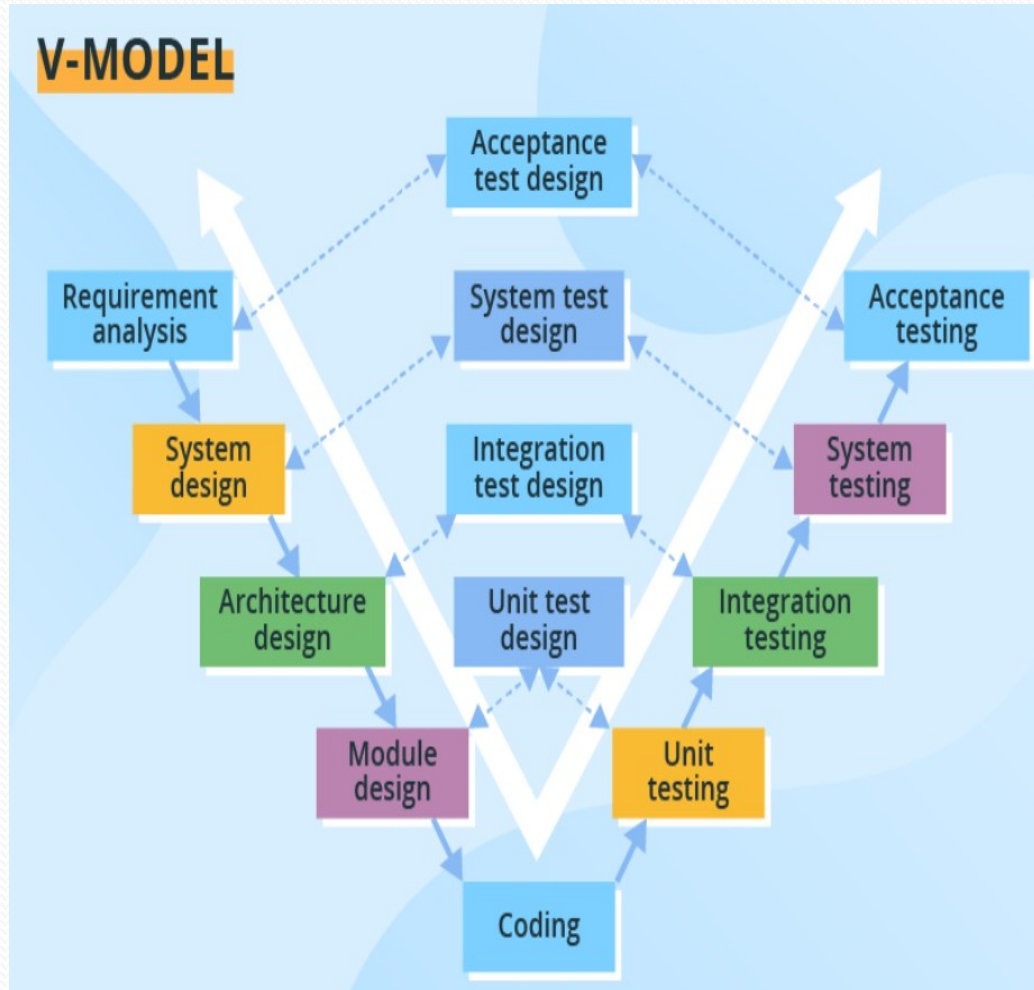
Waterfall model describes a staged development process

- **Based on hardware engineering models**
- **Change during development is unlikely**
- **Widely used in large systems: military and aerospace industries**

# When to use the Waterfall Model

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

## 2. V-Shaped SDLC Model



- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

# V-Shaped Steps

- **Project and Requirements Planning** – allocate resources
- **Product Requirements and Specification Analysis** – complete specification of the software system
- **Architecture or High-Level Design** – defines how software functions fulfill the design
- **Detailed Design** – develop algorithms for each architectural component
- **Production, operation and maintenance** – provide for enhancement and corrections
- **System and acceptance testing** – check the entire software system in its environment
- **Integration and Testing** – check that modules interconnect correctly
- **Unit testing** – check that each module acts as expected
- **Coding** – transform algorithms into software

# V-Shaped Strengths

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

# V-Shaped Weaknesses

- Does not easily handle **concurrent events**
- Does not handle **iterations** or phases
- Does not easily handle **dynamic changes in requirements**
- Does not contain **risk analysis** activities

# When to use the V-Shaped Model

- Excellent choice for **systems requiring high reliability** – hospital patient control applications
- **All requirements are known** up-front
- When it can be modified to **handle changing requirements beyond analysis phase**
- **Solution and technology are known**

### 3. Structured Evolutionary Prototyping Model

- Developers build a prototype during the requirements phase
- Prototype is evaluated by end users
- Users give corrective feedback
- Developers further refine the prototype
- When the user is satisfied, the prototype code is brought up to the standards needed for a final product.

# Structured Evolutionary Prototyping Steps

- A preliminary project plan is developed
- An partial high-level paper model is created
- The model is source for a partial requirements specification
- A prototype is built with basic and critical attributes
- The designer builds
  - the database
  - user interface
  - algorithmic functions
- The designer demonstrates the prototype, the user evaluates for problems and suggests improvements.
- This loop continues until the user is satisfied

# Structured Evolutionary Prototyping

## Strengths

- Customers can “see” the system requirements as they are being gathered
- Developers learn from customers
- A more accurate end product
- Unexpected requirements accommodated
- Allows for flexible design and development
- Steady, visible signs of progress produced
- Interaction with the prototype stimulates awareness of additional needed functionality

# Structured Evolutionary Prototyping

## Weaknesses

- Tendency to abandon structured program development for “code-and-fix” development
- Bad reputation for “quick-and-dirty” methods
- Overall maintainability may be overlooked
- The customer may want the prototype delivered.
- Process may continue forever (scope creep)

# When to use Structured Evolutionary Prototyping

- Requirements are unstable or have to be clarified
- As the requirements clarification stage of a waterfall model
- Develop user interfaces
- Short-lived demonstrations
- New, original development
- With the analysis and design portions of object-oriented development.

## 4. Rapid Application Model (RAD)

- **Requirements planning phase** (a workshop utilizing structured discussion of business problems)
- **User description phase** – automated tools capture information from users
- **Construction phase** – productivity tools, such as code generators, screen generators, etc. inside a time-box. (“Do until done”)
- **Cutover phase** -- installation of the system, user acceptance testing and user training

# RAD Strengths

- **Reduced cycle time** and improved productivity with fewer people means lower costs
- **Time-box** approach mitigates cost and schedule risk
- **Customer involved throughout** the complete cycle minimizes risk of not achieving customer satisfaction and business needs
- Focus moves from documentation to code.
- **Uses modeling concepts** to capture information about business, data, and processes.

# RAD Weaknesses

- Accelerated development process **must give quick responses** to the user
- Risk of **never achieving closure**
- Hard to use with **legacy systems**
- Requires a system that can be **modularized**
- Developers and customers must be **committed to rapid-fire activities** in an abbreviated time frame.

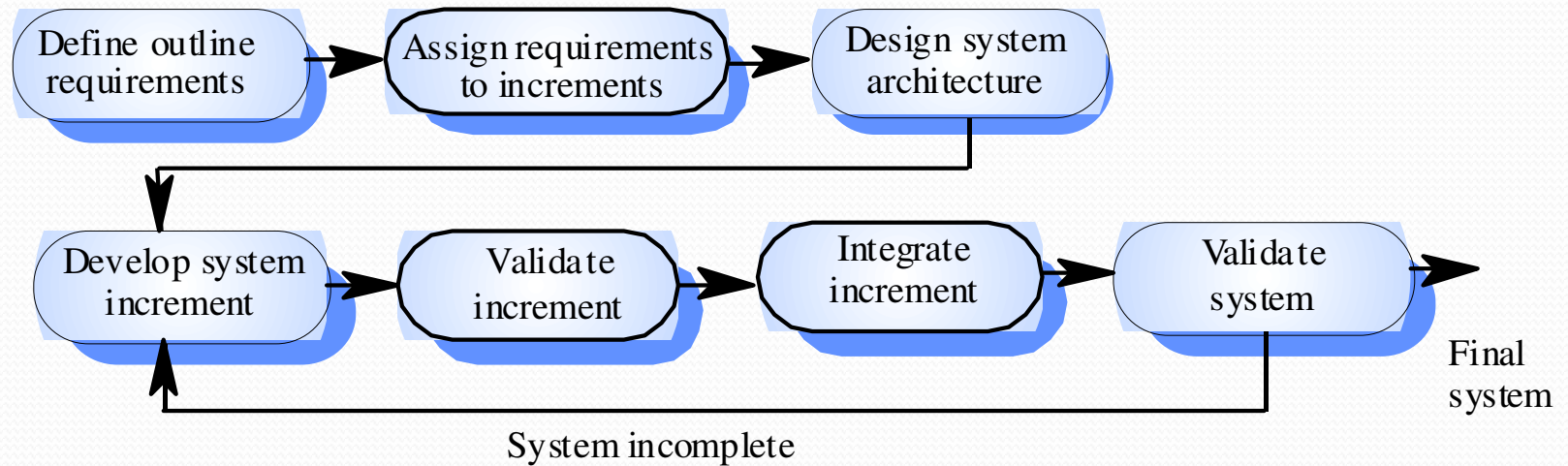
# When to use RAD

- Reasonably **well-known** requirements
- User involved **throughout** the life cycle
- Project can be **time-boxed**
- Functionality delivered in **increments**
- **High performance** not required
- **Low technical risks**
- System **can be modularized**

## 5. Incremental Development

- Rather than deliver the system as a single delivery, **the development and delivery is broken down into increments** with each increment delivering part of the required functionality.
- **User requirements are prioritised** and **the highest priority requirements are included in early increments.**
- **Once the development of an increment is started, the requirements are frozen** though requirements for later increments can continue to evolve.

# Incremental Development



## Incremental Development –Advantages

- **Customer value** can be delivered with each increment so system functionality is available earlier.
- **Early increments** act as a prototype to help elicit requirements for later increments.
- **Lower risk of overall project failure.**
- **The highest priority system services**  
tend to receive the most testing.

# Incremental Model Strengths

- Each release delivers an **operational product**
- Customer can **respond to each build**
- Uses “divide and conquer” **breakdown of tasks**
- Lowers **initial delivery cost**
- Initial **product delivery is faster**
- Customers get **important functionality early**
- Risk of **changing requirements is reduced**

# Incremental Development – Problems

- Lack of process visibility.
- Systems are often poorly structured.

- **Applicability claims in the literature:**

- For small or medium-size interactive systems.
- For parts of large systems (e.g. the user interface).
- For short-lifetime systems.

# When to use the Incremental Model

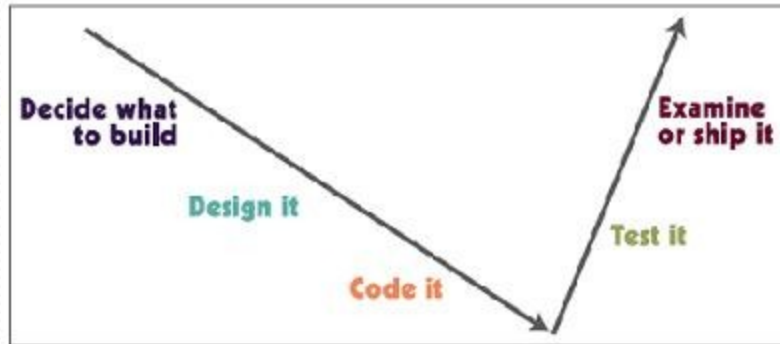
- Risk, funding, schedule, program complexity, or need for **early realization of benefits.**
- Most of the requirements are known upfront but are expected to **evolve over time**
- A need to **get basic functionality to the market early**
- On projects which have **lengthy development schedules**
- On a project with **new technology**

# Incremental means adding, iterative means reworking (by Alistair Cockburn)

- **Incremental development** is a staging and scheduling strategy in which the various parts of the system are developed at different times or rates and integrated as they are completed. The alternative is to develop the entire system with a big bang integration at the end.
- **Iterative development** is a rework scheduling strategy in which time is set aside to revise and improve parts of the system. The alternative development is to get it right the first time (or at least declare that it is right!).

| Increment                                                   | Iterate                                                        |
|-------------------------------------------------------------|----------------------------------------------------------------|
| fundamentally means<br>."add onto"                          | fundamentally means<br>."change"                               |
| repeating the process<br>on a <i>new</i> section of<br>work | repeating the process<br>on the <i>same</i> section of<br>work |
| repeat the process (<br>design, implement,<br>evaluate)     | repeat the process (<br>design, implement,<br>evaluate)        |

# Incremental Development



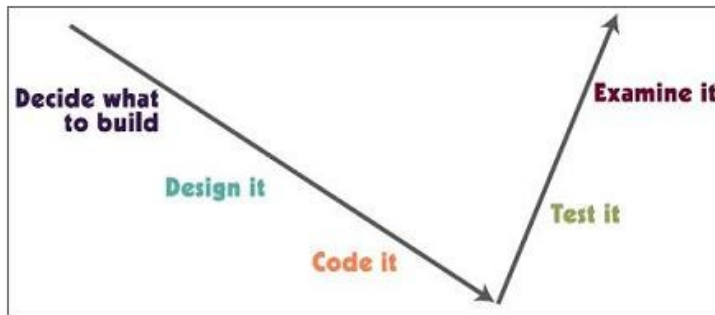
- The first increment delivers one slice of functionality through the whole system
- The second increment delivers another slice of functionality through the whole system
- The third increment delivers the rest of the system

|            | Feature 1 | Feature 2 | Feature 3 |
|------------|-----------|-----------|-----------|
| UI layer   |           |           |           |
| App layer  |           |           |           |
| Middleware |           |           |           |
| Database   |           |           |           |

|            | Feature 1 | Feature 2 | Feature 3 |
|------------|-----------|-----------|-----------|
| UI layer   |           |           |           |
| App layer  |           |           |           |
| Middleware |           |           |           |
| Database   |           |           |           |

|            | Feature 1 | Feature 2 | Feature 3 |
|------------|-----------|-----------|-----------|
| UI layer   |           |           |           |
| App layer  |           |           |           |
| Middleware |           |           |           |
| Database   |           |           |           |

# Iterative Development



- The first iteration delivers enough of feature 1 to evaluate what is correct and what needs revision.
- After the second iteration, some revised parts still need improvement.
- The third iteration produces the final and stable feature

|            | Feature 1 | Feature 2 |
|------------|-----------|-----------|
| UI layer   | ?         |           |
| App layer  | ?         |           |
| Middleware | ✓         |           |
| Database   | ?         |           |

|            | Feature 1 | Feature 2 |
|------------|-----------|-----------|
| UI layer   | ??        |           |
| App layer  | ??        |           |
| Middleware | ✓         |           |
| Database   | ✓         |           |

|            | Feature 1 | Feature 2 |
|------------|-----------|-----------|
| UI layer   | ✓         |           |
| App layer  | ✓         |           |
| Middleware | ✓         |           |
| Database   | ✓         |           |

# Problems with incremental development (from a waterfall eye...)

- **Management problems**
  - Progress can be hard to judge and problems hard to find because there is no documentation to demonstrate what has been done.
- **Contractual problems**
  - The normal contract may include a specification; without a specification, different forms of contract have to be used.
- **Validation problems**
  - Without a specification, what is the system being tested against?
- **Maintenance problems**
  - Continual change tends to corrupt software structure making it more expensive to change and evolve to meet new requirements.

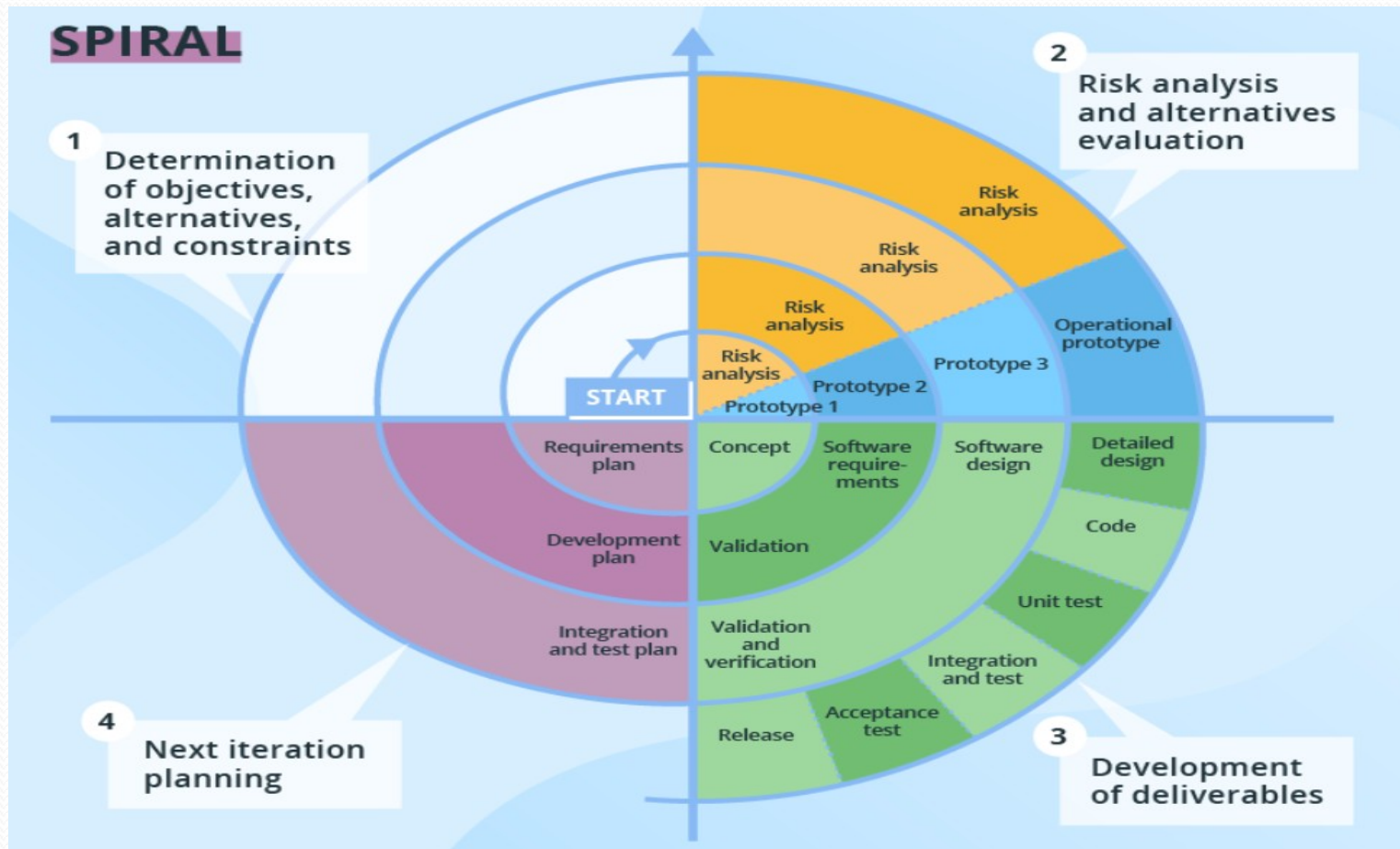
# Incremental & iterative - summary

- Iterative model - This model iterates requirements, design, build and test phases again and again for each requirement and builds up a system iteratively till the system is completely build.
- Incremental model - It is non integrated development model. This model divides work in chunks and one team can work on many chunks. It is more flexible.
- Spiral model - This model uses series of prototypes in stages, the development ends when the prototypes are developed into functional system. It is flexible model and used for large and complicated projects.
- Evolutionary model - It is more customer focused model. In this model the software is divided in small units which is delivered earlier to the customers.
- V-Model - It is more focused on testing. For every phase some testing activity are done.

## 6.Spiral Development

- **Process is conceived as a spiral** rather than as a sequence of activities with backtracking.
- **Each loop in the spiral represents a phase** in the process.
- **No fixed phases such as specification or design** - loops in the spiral are chosen depending on what is required.
- **Risks are explicitly assessed and resolved** throughout the process.

# Spiral Model



# Spiral model sectors

- **Objective setting**

- Specific objectives for the phase are identified.

- **Risk assessment and reduction**

- Risks are assessed and activities put in place to reduce the key risks.

- **Development and validation**

- A development model for the system is chosen which can be any of the generic models.

- **Planning**

- The project is reviewed and the next phase of the spiral is planned.

# Spiral Model Strengths

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users
- Cumulative costs assessed frequently

# Spiral Model Weaknesses

- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

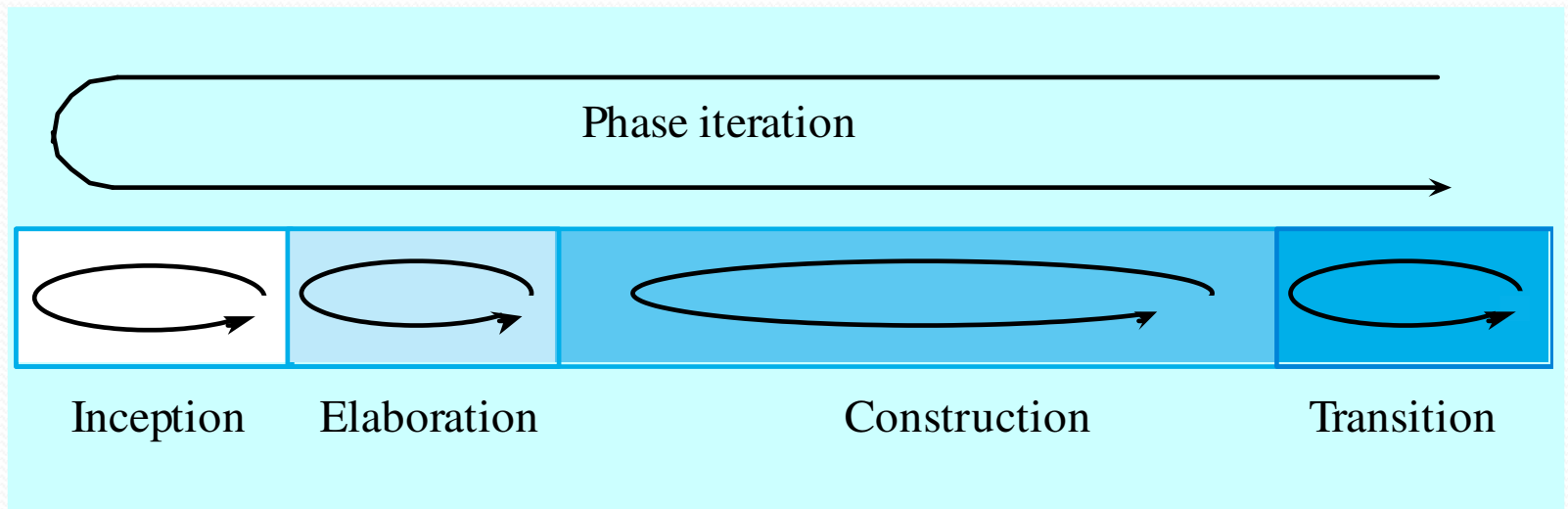
# When to use Spiral Model

- When creation of a prototype is appropriate
- 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
- New product line
- Significant changes are expected (research and exploration)

## 7. The (Rational) Unified Process

- A modern process model derived from the work on the UML.
- Normally described from 3 perspectives
  - A **dynamic** perspective that shows phases over time;
  - A **static** perspective that shows process activities;
  - A **practice** perspective that suggests good practice.

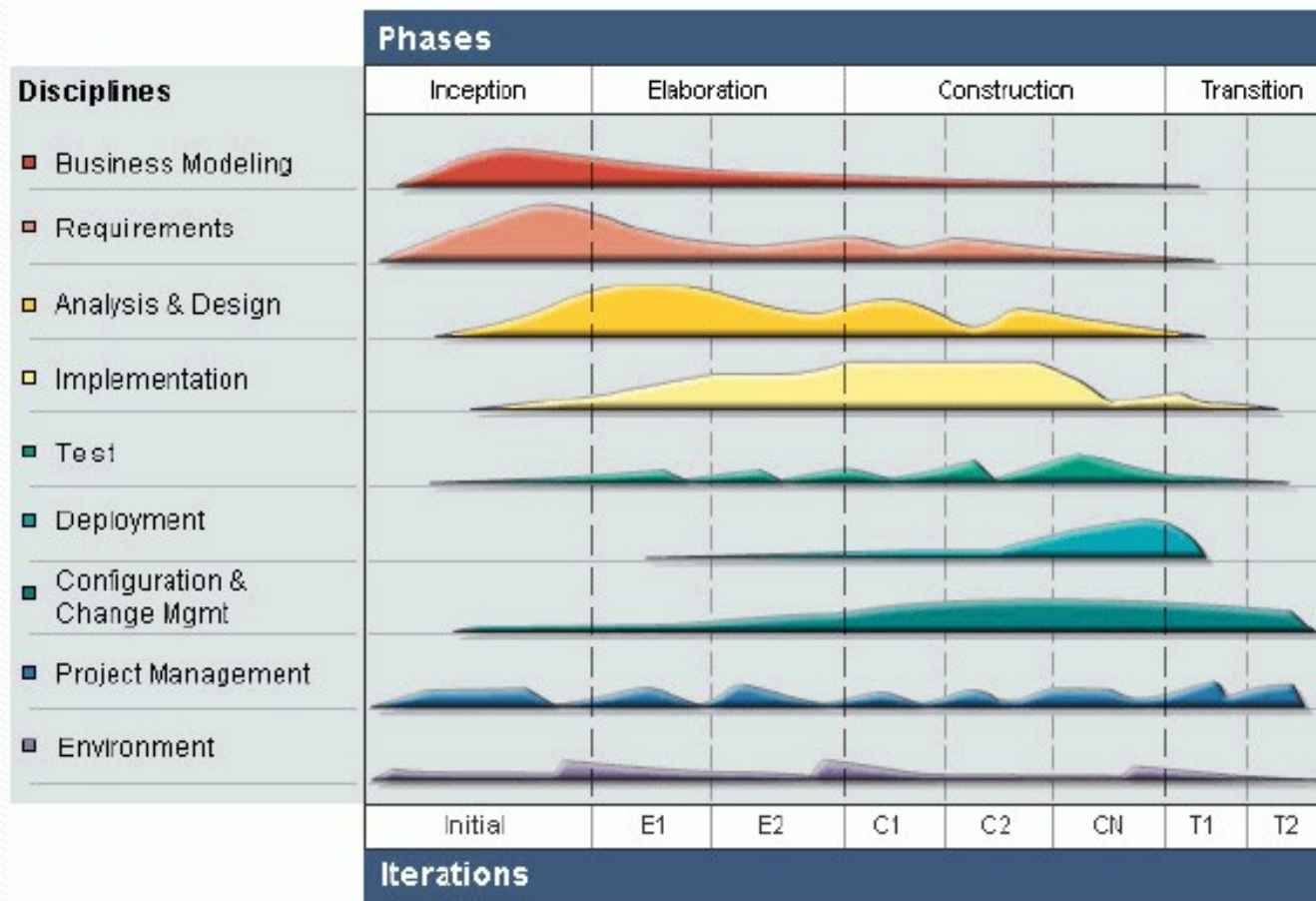
# (R)UP phase model



## (R)UP phases

- One cycle consists of four phases:
  - **Inception**
    - Establish the business case for the system.
  - **Elaboration**
    - Develop an understanding of the problem domain and the system architecture.
  - **Construction**
    - System design, programming and testing.
  - **Transition**
    - Deploy the system in its operating environment.

# (R)UP phases and iterations



Picture taken from:

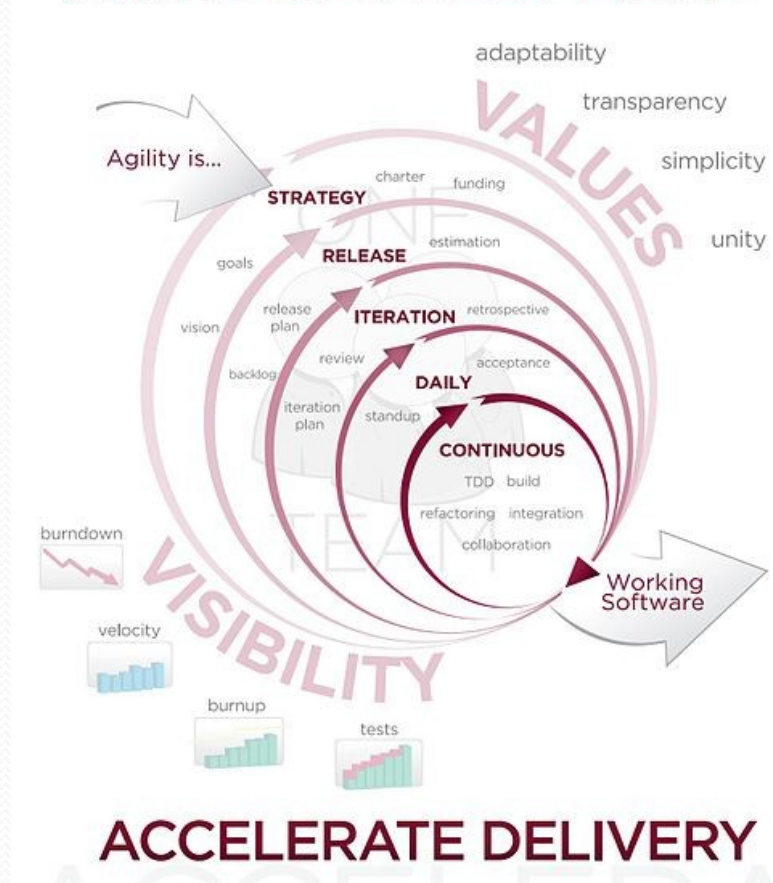
<http://www.ibm.com/developerworks/webservices/library/ws-soa-term2/>

## (R)UP good practice

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software

# 8. Agile Development

## AGILE DEVELOPMENT



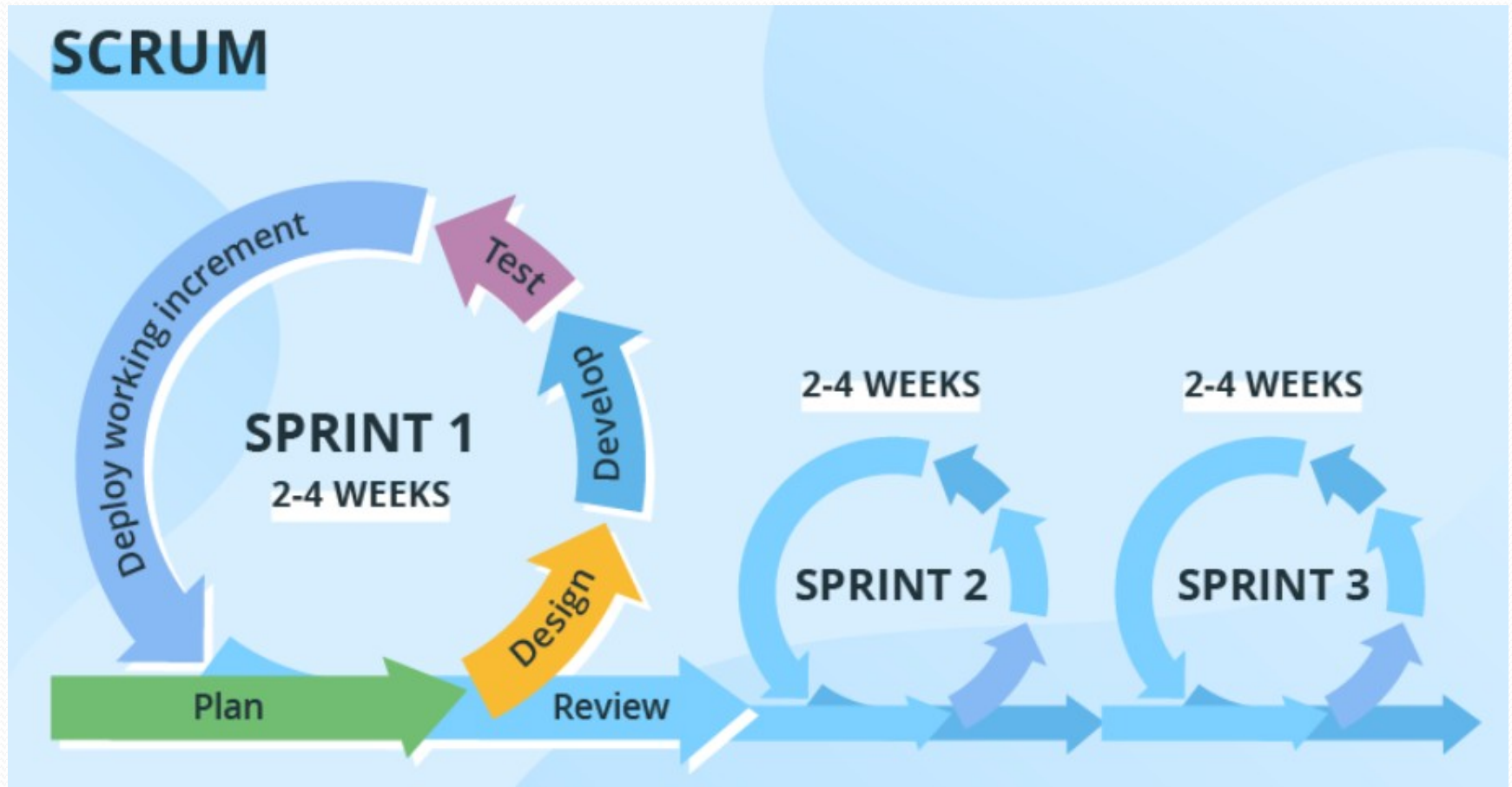
# Project Failure – the trigger for Agility

- One of the primary causes of project failure was the **extended period of time it took to develop a system.**
- Costs escalated and requirements changed.
- Agile methods intend to **develop systems more quickly with limited time spent on analysis and design.**

# What is an Agile method? (1)

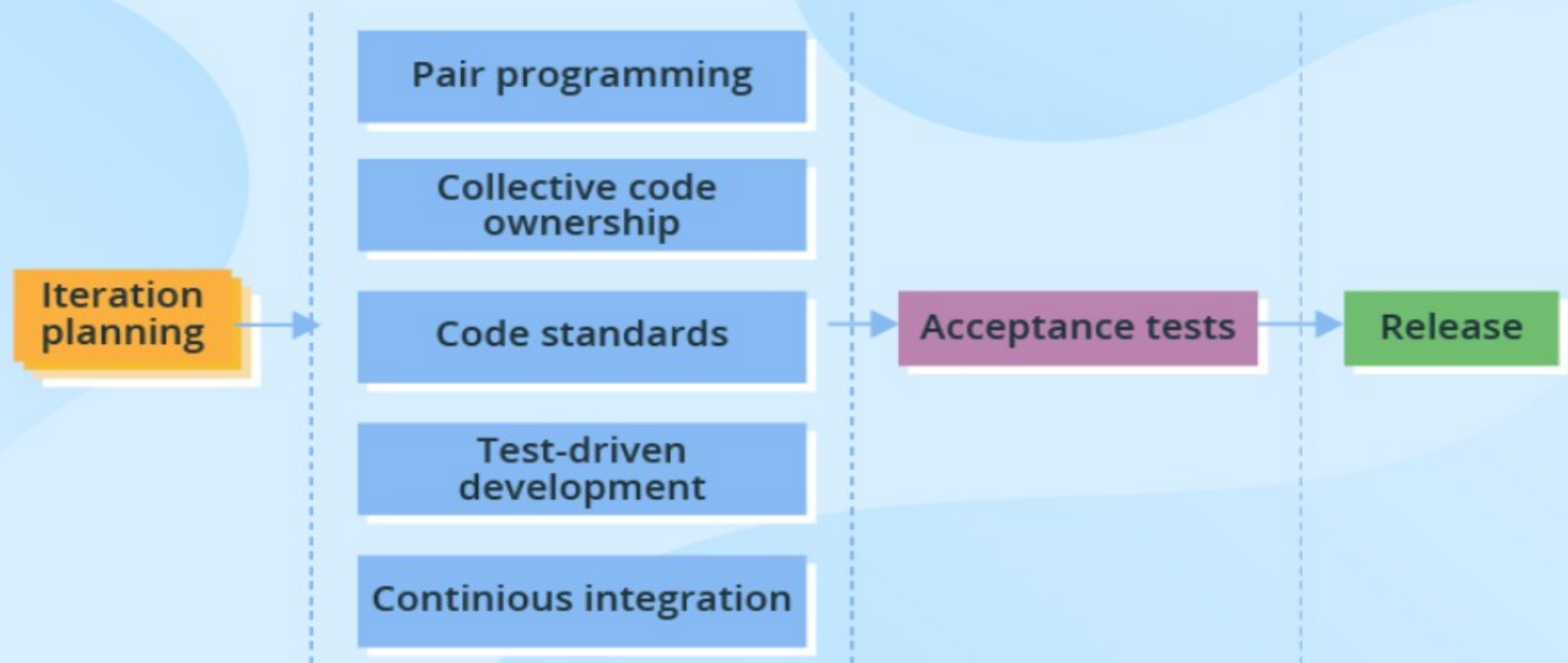
- **Focus on the code rather than the design.**
- **Based on an iterative approach to software development.**
- **Intended to deliver working software quickly.**
- Evolve quickly to **meet changing requirements.**
- There are claims that agile methods are probably best suited to small/medium-sized business systems or PC products.
- Most popular Agile methods are scrum , extreme programming and KANBAN

# Scrum



# Extreme Programming

## EXTREME PROGRAMMING (XP)



# KANBAN

## KANBAN



# Summary of Principles of Agile Methods

| Principle            | Description                                                                                                                                                                         |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Customer involvement | Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system. |
| Incremental delivery | The software is developed in increments with the customer specifying the requirements to be included in each increment.                                                             |
| People not process   | The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.             |
| Embrace change       | Expect the system requirements to change and so design the system to accommodate these changes.                                                                                     |
| Maintain simplicity  | Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.                  |

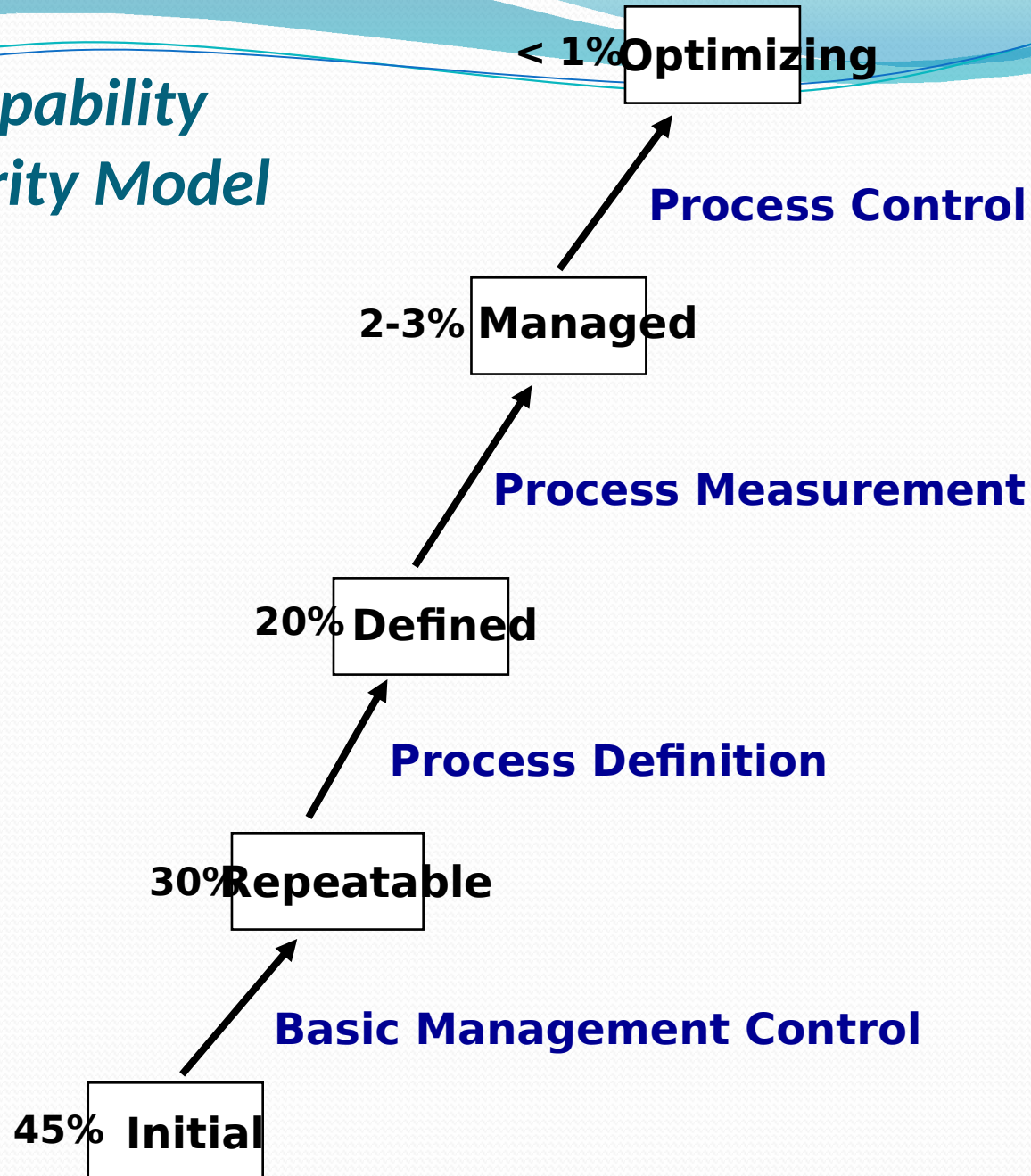
# When to use Agile method

- When you have best programmers
- Full participation of User and decision makers
- At least one experienced and competent staff
- rapid feedback on decisions from user and decision maker
- The system is not complex
- The design is flexible whenever the requirements are change

# Process Assessment

- In the CMM model, the maturity level of an organization tells us to what extent an organization can produce low cost, high quality software.
- Having known the current maturity level, an organization can work to reach the next higher level.

# *SEI Capability Maturity Model*



# CMM - Initial (Level 1)

- ▮ *The software process is characterized as ad hoc, occasionally even chaotic*
- ▮ *Few processes are defined*
- ▮ *Success depends on individual effort and heroics*

***“BASICALLY NO CONTROL”***

# **CMM - Repeatable (Level 2)**

- ***Basic project management processes are established to track cost, schedule, and functionality***
- ***The necessary process discipline is in place to repeat earlier successes on projects with similar applications***
- ***Success achieved through basic project management; not advanced technologies***

***“BASIC MANAGEMENT CONTROL”***

# CMM - Defined (Level 3)

- ▢ *The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization*
- ▢ *All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software*
- ▢ *Formality lends itself to improvement*

***“PROCESS DEFINITION”***

# CMM - Managed (Level 4)

- ▮ *Detailed measures of the software process and product quality are collected*
- ▮ *Both the software process and products are quantitatively understood and controlled*
- ▮ *A software metrics program is in use*

***“PROCESS MEASUREMENT”***

# CMM - Optimizing (Level 5)

□ ***Continuous process improvement is enabled by quantitative (metrics) feedback from the process***

□ ***Continuous process improvement is enabled by piloting innovative ideas***  
***“PROCESS CONTROL”***

***and technologies***



**Q?**

**Thank You!**

# Quiz-1(10 %)

1. What is the purpose of process model
  2. Briefly Describe the difference between computer science and Software Engineering
- Select the best process model for the following system and justify your answer
- 1- developed online sale and payment system
  - 2- develop E-commerce system
  - 3- Crime Notification system