Software Engineering

(Agile Software Engineering:
Agile methods, Scrum, and Extreme Programming)

Course Learning Outcome

CLO1: Recognize Software Engineering principles, **life cycle phases, processes**, and activities.

The Software Process

- Software Process is a structured set of activities required to develop a software system.
- There are many different software processes, but all involve:
  - **Specification –** defining what the system should do;
  - **Design and implementation** – defining the organization of the system and implementing the system;
  - **Validation** – checking that it does what the customer wants;
  - **Evolution** – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Software Process Descriptions

- When we describe and discuss processes, we usually talk about the activities or **actions** in these processes such as specifying a *data model*, *designing a user interface*, *etc. and the ordering of these activities*.
- Process descriptions may also include:
  - Products, which are the outcomes of a process activity;
  - Roles, which reflect the responsibilities of the people involved in the process;
  - Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

Plan-Driven and Agile Processes

- **Plan-driven (Prescriptive) processes** are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In **Agile processes**, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

Traditional/Planned Software Process Models

Software Process Models

- **The Waterfall Process Model**

  Plan-driven model. Separate and distinct phases of specification and development.

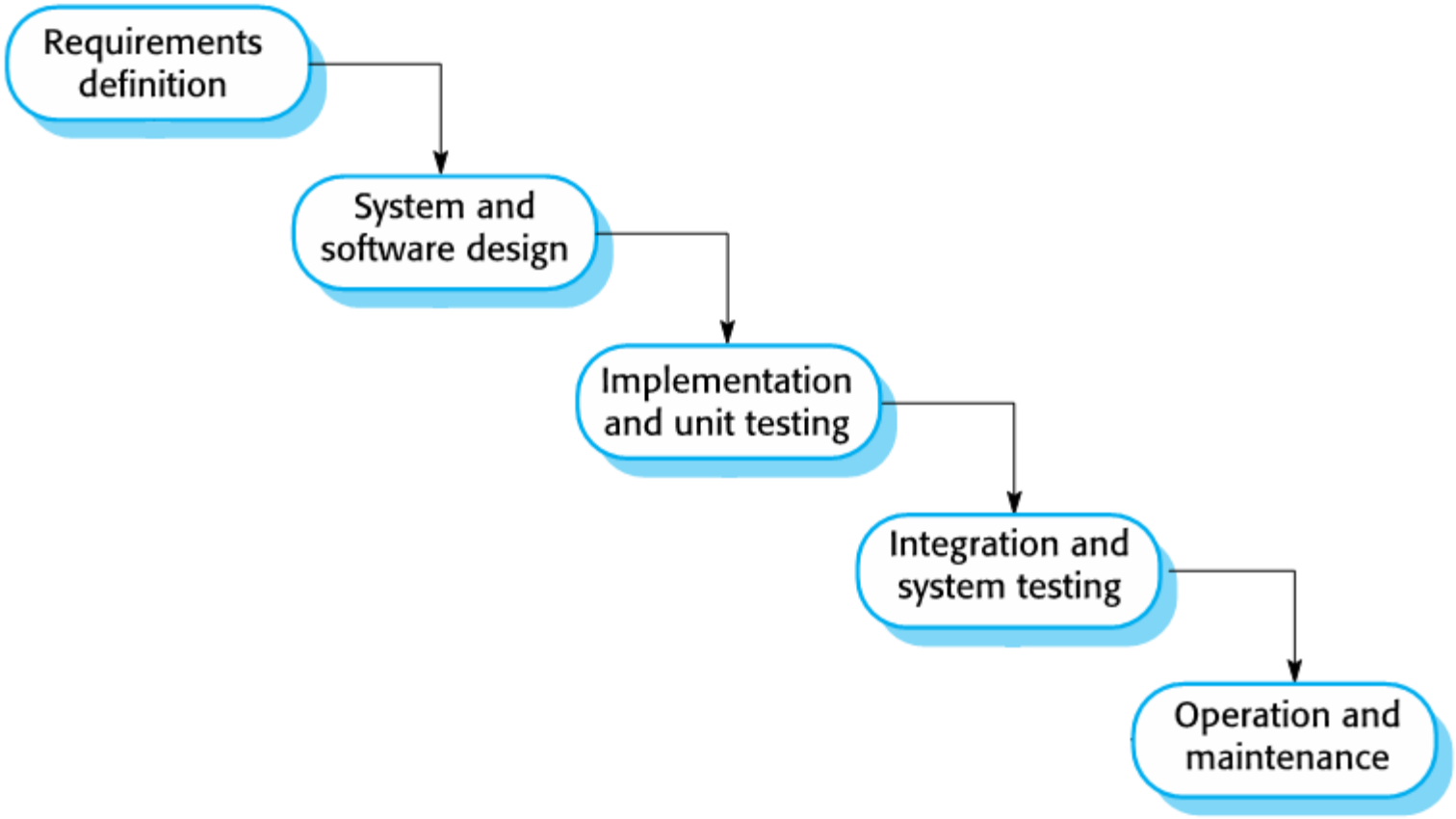
- **Incremental Process Model**

  Plan-driven model. Separate and distinct phases of specification and development in increments.

- **Evolutionary/Iterative Process Model**

  An iterative model. It enables you to develop increasingly more complete version of the software. Emphasis on risk management.

The waterfall model



Waterfall model phases

- There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- Each major phase is marked by milestones and deliverables (artifacts)

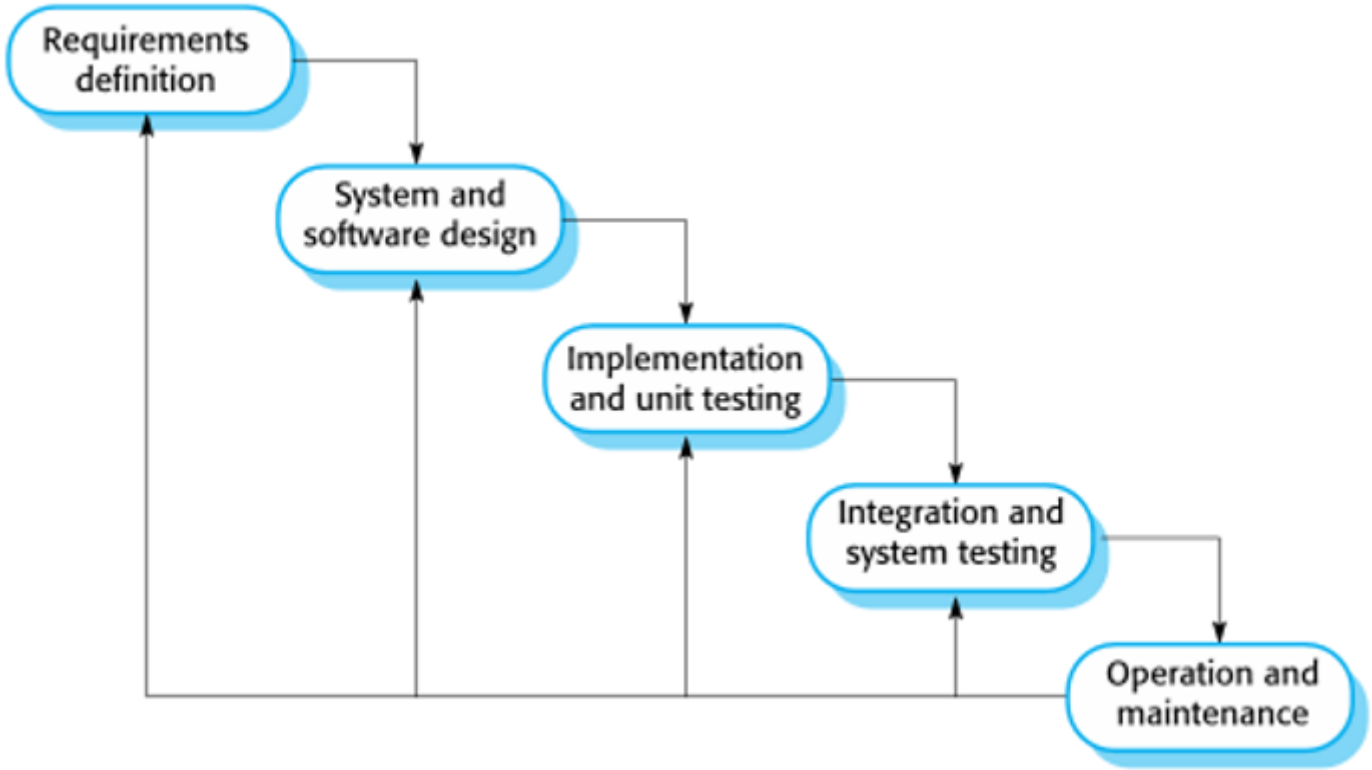
When to Use Waterfall Model

1. It Works for **well understood** problems with minimal or no changes in the requirements such as governmental projects.
2. Simple and easy to explain to customers.

Waterfall Model drawback

1. The main **drawback** of the waterfall model is the **difficulty of accommodating change** after the process is underway. In principle, a phase has to be complete before moving onto the next phase.
2. It is often difficult for the customer to state all requirements explicitly. The **Inflexible partitioning** of the project into distinct stages makes it difficult to respond to changing customer requirements.
3. Therefore, this model is only appropriate when the **requirements are well-understood** and changes will be fairly limited during the design process.

Modified Waterfall

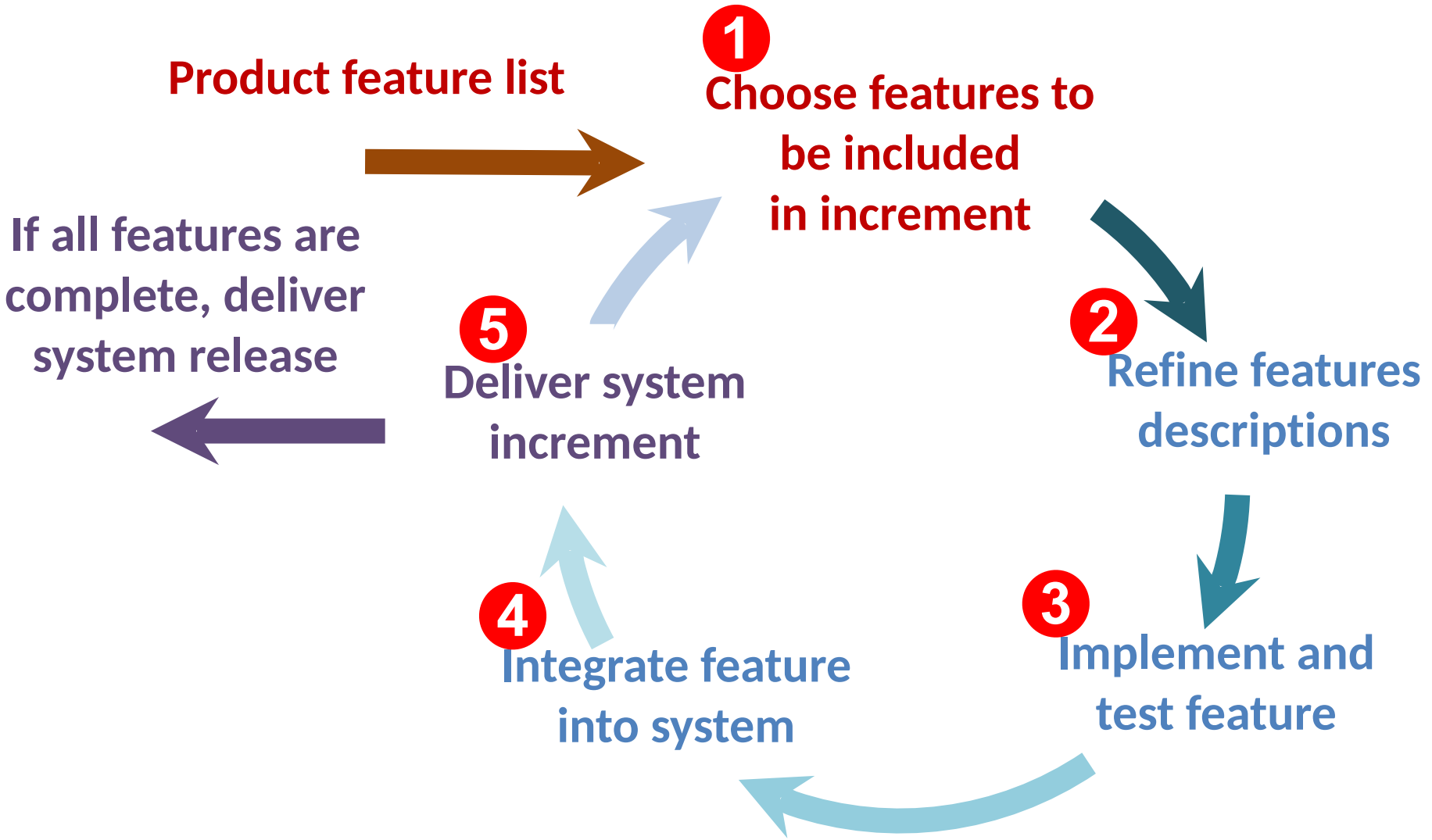


The model has been modified to allow going back to previous phases and this opened the door for other iterative models

Incremental Model

- Rather than deliver the System as single delivery, the system is broken down into increments with each increment adding a part of the required functionality.
- The Incremental model combines the elements of linear and parallel process flow.
- The incremental process model focuses on the delivery of an operational product with each increment.
- User requirements are prioritised then the highest priority requirement 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 Model



Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Incremental Model Activities

1. **Choose features to be included in an increment**
   Using the list of features in the planned product, select those features that can be implemented in the next product increment.
2. **Refine feature descriptions**
   Add detail to the feature descriptions so that the team has a common understanding of each feature and there is sufficient detail to begin implementation.
3. **Implement and test**
   Implement the feature and develop automated tests for that feature, tests that will validate that the system's behaviour is consistent with its description.
4. **Integrate feature and test**
   Integrate the developed feature with the existing system and test it to check that it works in conjunction with other features.
5. **Deliver system increment**
   Deliver the system increment to product manager for checking and comments. If enough features have been implemented, release a version of the system for customer use.

Source: Ian Sommerville (2019), Engineering Software Products: An Introduction to Modern Software Engineering, Pearson.

Incremental Model Benefits

- The cost of accommodating changing customer requirements is reduced.
- It is easier to get customer feedback on the development work that has been done.
- More rapid delivery and deployment of useful software to the customer is possible.
- The highest Priority system services tends to receive most system testing.
- Reduce risk of overall project failure (compared to waterfall).

Incremental Model Problems

➢ **System structure tends to degrade as new increments are added.**

Unless time and money is spent on **refactoring** to improve the software, regular change tends to corrupt its structure.

- **Difference with evolutionary models**: The list of features is defined from the beginning in the classical incremental model.

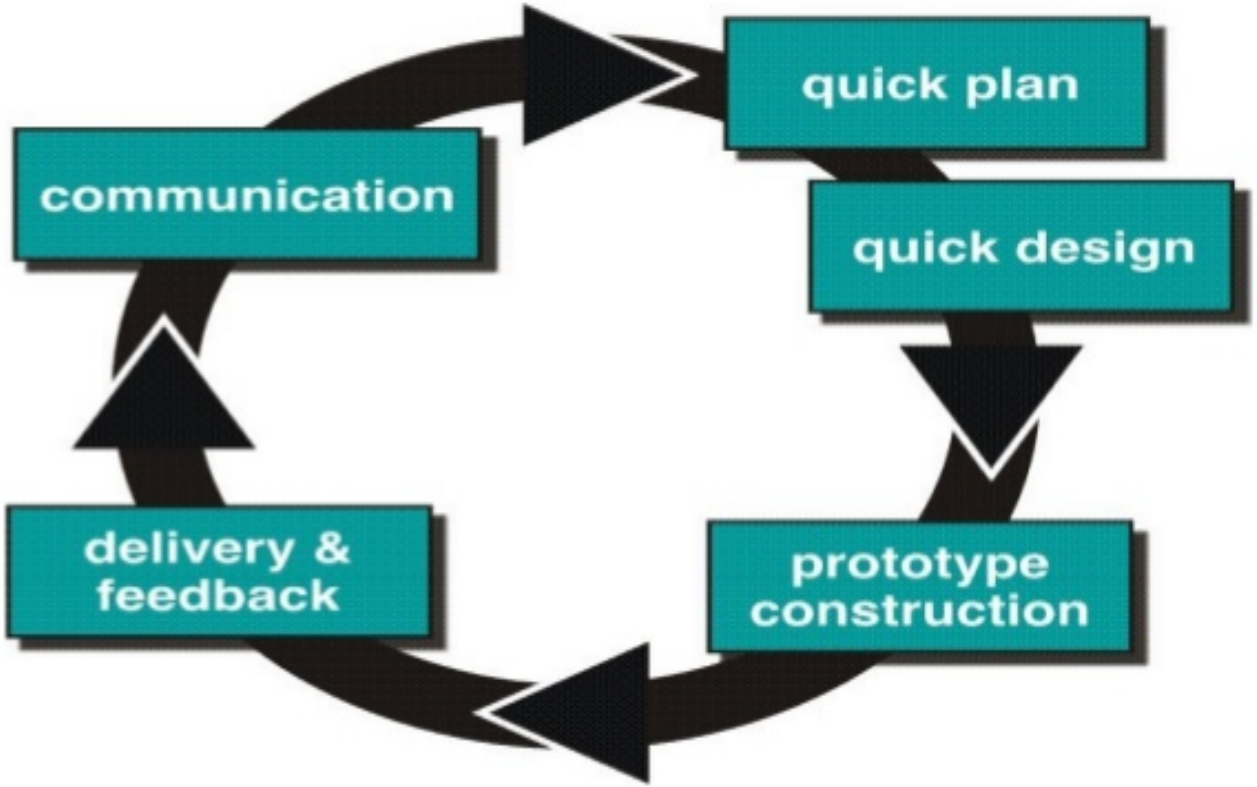
Evolutionary Model

- The Evolutionary models are iterative. They are characterized in a manner that enables you to develop increasingly more complete versions of the software in increments.
- It is used when core or system requirements are well understood but **additional system requirements** have yet to be defined.
- Here we present two **common evolutionary process models.**
  - Prototyping Model
  - Spiral Model

Evolutionary Models - Prototyping

- Often a customer defines a set of general objectives for software, but doesn't identify details for the functions and features. In other cases, developers may be unsure of the efficiency of an algorithm. In these cases Prototyping may be the best approach.
- The Prototype can serve as "the first system" Although some prototypes are built as "Throw away", others are evolutionary in the sense that the prototype slowly evolves into the actual system.
- Users get a feel for the actual system and developers get to build something immediately.

Evolutionary Models - Prototyping



Evolutionary Models - Prototyping

- The prototyping paradigm begins with communication. You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- A prototyping iteration is planned quickly, and modelling ("quick design") occurs (e.g., human interface layout or output display formats).
- The quick design leads to the construction of a prototype. The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.
- Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better understand what needs to be done.

Evolutionary Models - Prototyping

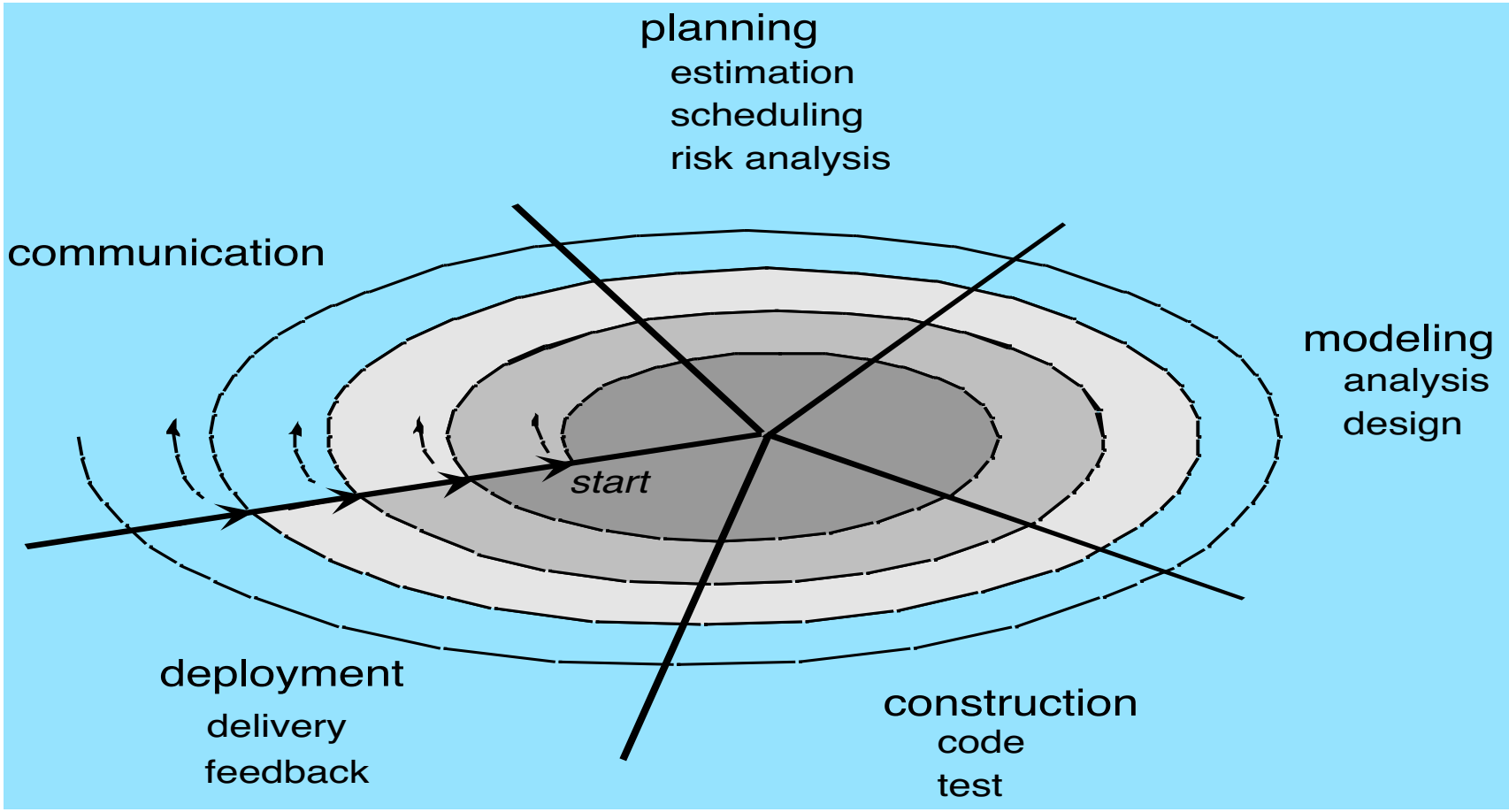
- **Benefits of Prototyping**
  Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately and improve the system before deploying the system.

- **Drawbacks of Prototyping**
  Stakeholders see what appears to be a working version of the software, unaware that in the rush to get it working you haven't considered overall software quality or long-term maintainability. For example: developers may use inappropriate operating system, programming language, or inefficient algorithm.

Evolutionary Models - Spiral

- Spiral Model couples the iterative nature of prototyping with controlled and systematic aspects of waterfall.
- It provides the potential for rapid development of increasingly more complete versions of the software.
- The Spiral Model is a risk-driven process model generator. It has two main distinguishing features.
  1. It is a cyclic nature for incrementally growing a system's degree of definition and implementation while decreasing its degree of risk.
  2. It has a set of milestones for ensuring stakeholders commitment to feasible and mutually satisfactory system solution.

Evolutionary Models - Spiral



Evolutionary Models - Spiral

- The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- **Each pass** through the planning region results in **adjustments** to the project **plan.**
- Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- The spiral model uses **prototyping** as a risk reduction mechanism.

Specialized Process Model

1. Component-Based Development
2. Unified Process

Component-Based Development

- Based on software reuse where systems are integrated from existing components or application systems (sometimes called COTS -Commercial-off-the-shelf systems).
- Reused elements may be configured to adapt their behaviour and functionality to a user's requirements
- Reuse is now the standard approach for building many types of business system.

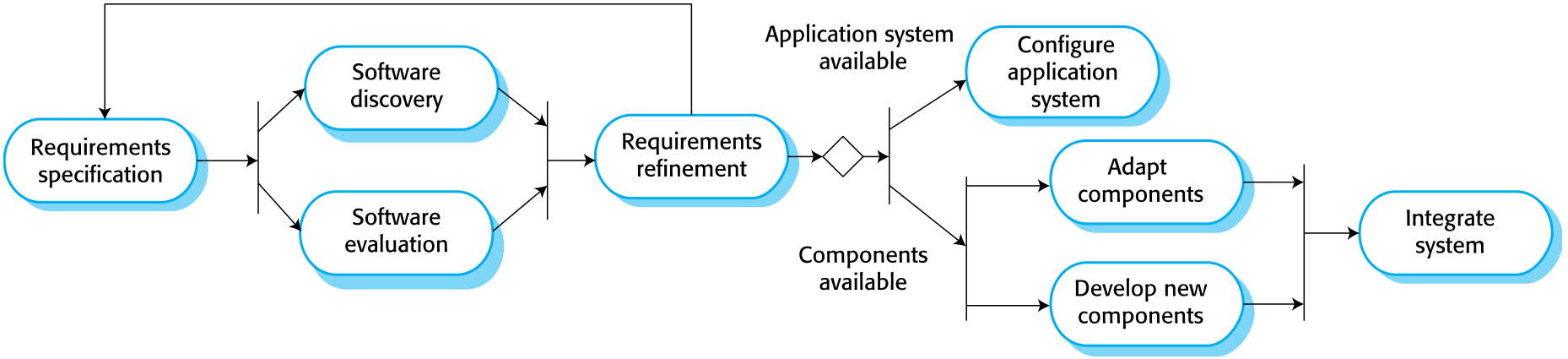
Component based development

- The process to apply when reuse is a development objective.
- You want to use components that are already made or you want to make components that may be **reused**.
- Reusability provides software engineers with a number of measurable benefits including a reduction in development time and a reduction in project cost if **component reuse** becomes part of your organization's culture

Types of reusable software

1. Stand-alone application systems that are configured for use in a particular environment.
2. Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
3. Web services that are developed according to service standards and which are available for remote invocation.

Reuse-oriented software engineering



Reusability Advantages and disadvantages

Advantages

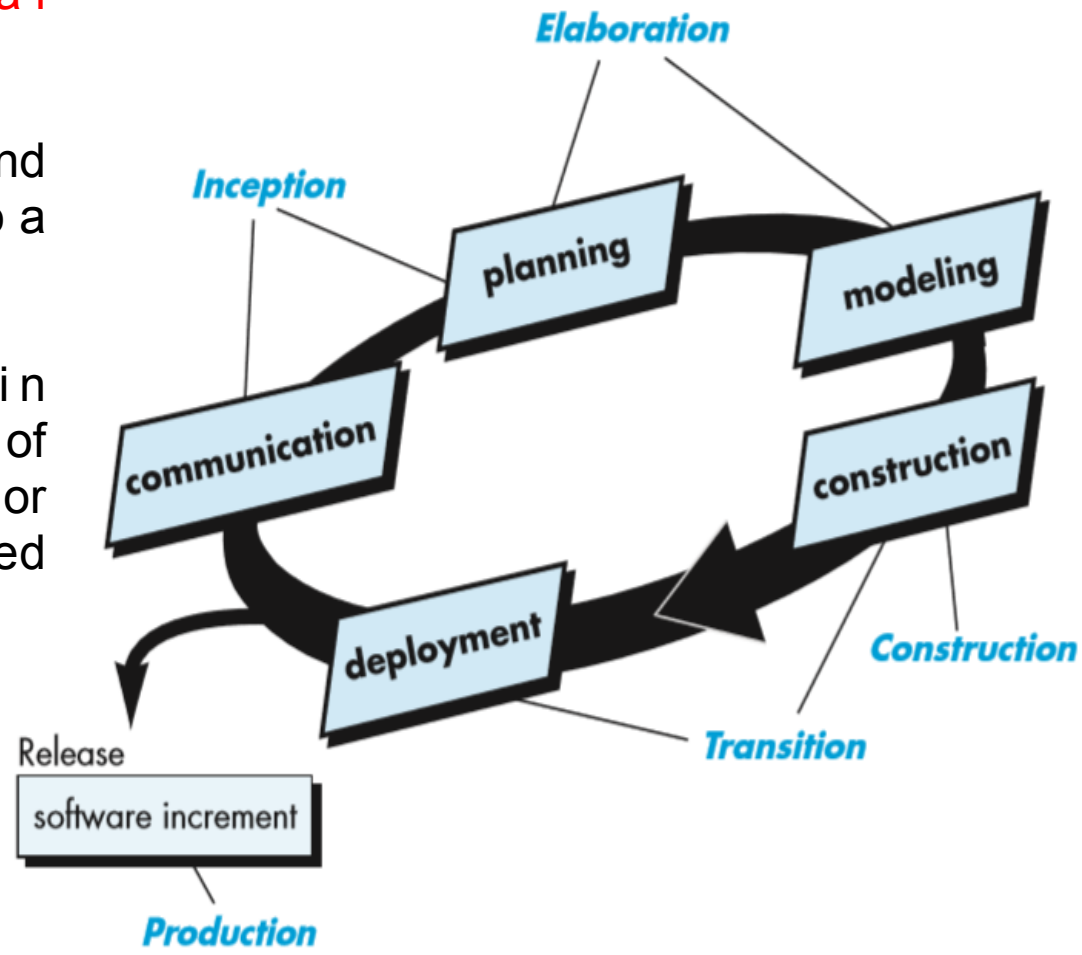
1. Reduced costs and risks as less software is developed from scratch
2. Faster delivery and deployment of system

Disadvantages

1. Requirements compromises are inevitable so system may not meet real needs of users
2. Loss of control over evolution of reused system elements

The Unified Process (UP)

- The Unified Process is an iterative and incremental development process.
- The Elaboration, Construction and Transition phases are divided into a series of timeboxed iterations.
- Each iteration results in an *increment*, which is a release of the system that contains added or improved functionality compared with the previous release.



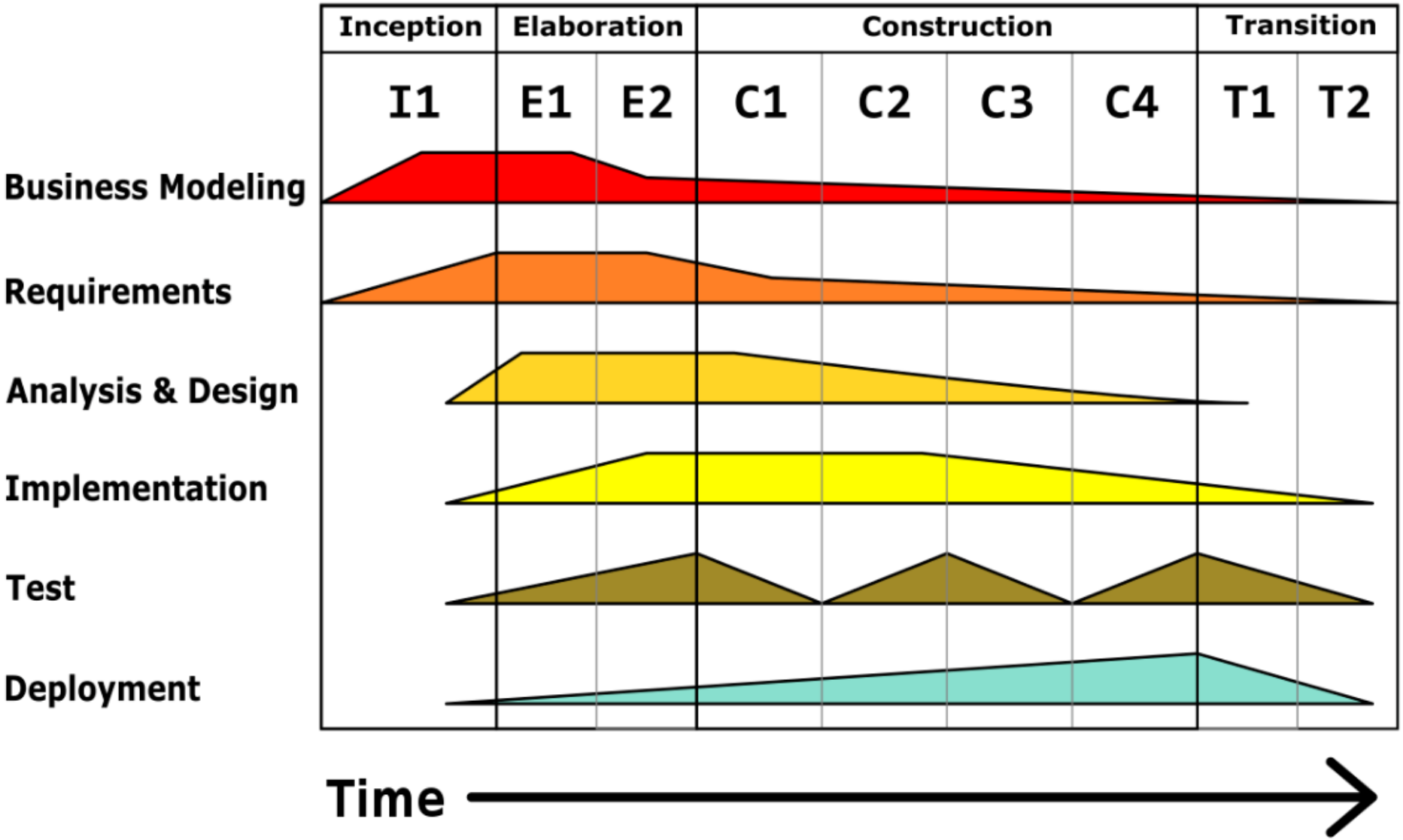
The Unified Process (UP)

- Phases of UP
  - Inception
    - Identify business requirements, described as *use-cases*
    - Propose a rough system architecture
  - Elaboration
    - Refine and expand the preliminary use-cases
    - Expand the architecture to include five different views
      - Use-case model, analysis model, design model, implementation model, deployment model

The Unified Process (UP)

- Construction
  - Develop or acquires the software components to accomplish an initial release
- Transition
  - Software is given to users for beta-testing to collect defects and necessary changes
  - Create support information, e.g., user manuals, installation procedures, trouble-shooting guides
- Production
  - Monitor the on-going use of the software, and evaluate defect reports and change requests

The Unified Process (UP)

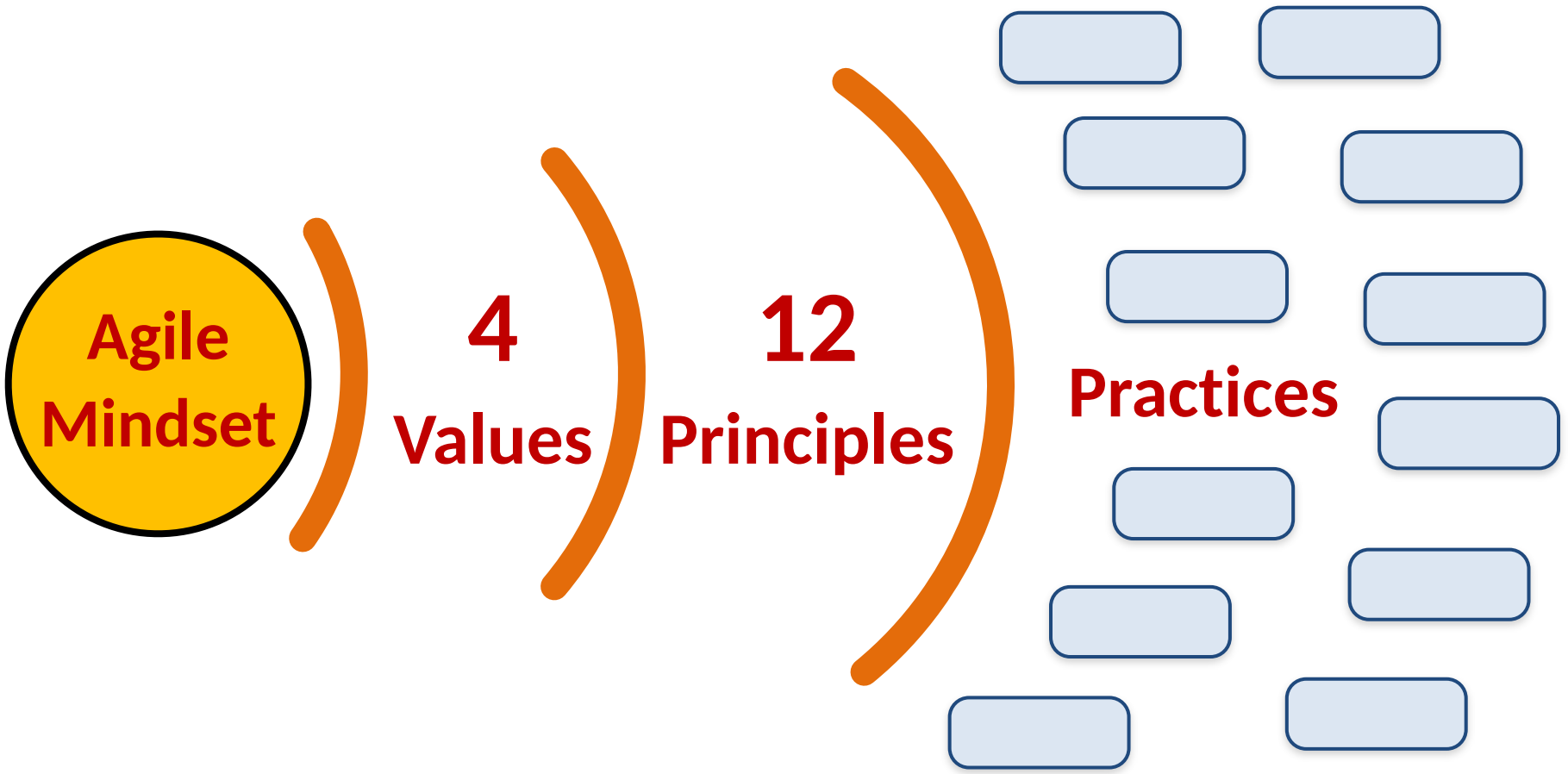


Agile Software Engineering

Agile software engineering

- Software products must be brought to market quickly so rapid software **development and delivery** is essential.
- Virtually all software products are now developed using an **agile approach** (except safety-critical systems: in direct opposition with agile manifesto and development principles).
- **Agile** software engineering focuses on **delivering** functionality **quickly**, **responding** to **changing** product specifications and **minimizing** development **overheads**.

Agile Manifesto Values, Principles, and Common Practices



Agile Manifesto and Mindset

Agile is a mindset defined by 4 values, guided by 12 principles, and manifested through many different practices.

Agile practitioners select practices based on their needs.

4
Agile Values

The Four Values of the Agile Manifesto

(Manifesto for Agile Software Development, 2001)

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

1. **individuals and interactions** over **processes and tools**
2. **working software** over **comprehensive documentation**
3. **customer collaboration** over **contract negotiation**
4. **responding to change** over **following a plan**

That is, while there is value in the items on the right, we value the items on the left more.

12
Agile Principles

The Twelve Principles Behind the Agile Manifesto

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. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support their needs 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.

The Twelve Principles Behind the Agile Manifesto

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.

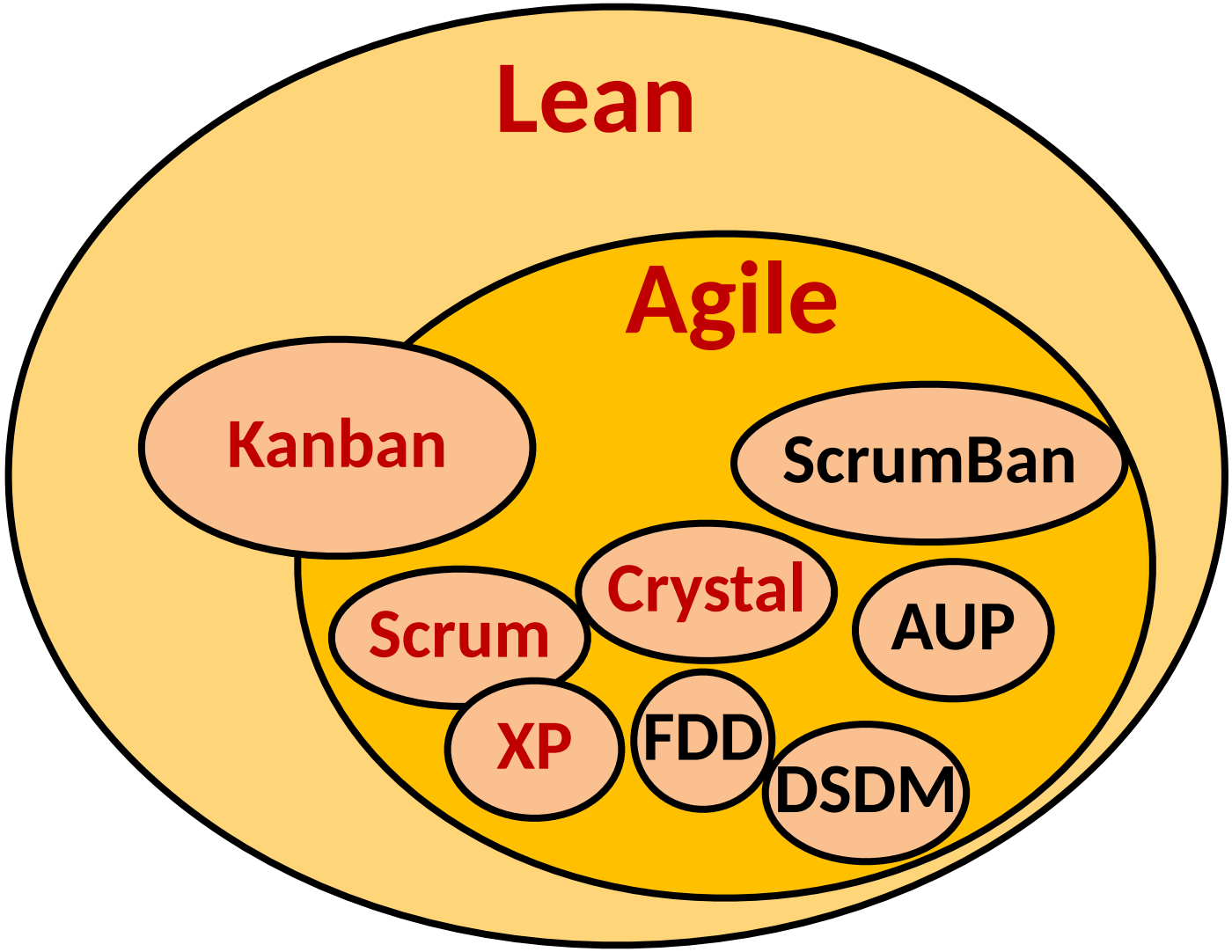
9. Continuous attention to technical excellence and good design enhances agility.

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.

Agile is a Blanket Term for Many Approaches



Key points

- Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- Traditional process models describe the organization of software processes.
  - Examples of these traditional models include the ‘waterfall’ model, incremental development
- Requirements engineering is the process of developing a software specification.

Key points

- Design and implementation processes are concerned with transforming a requirements specification into an executable software system.
- Software validation is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- Software evolution takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.
- Processes should include activities such as prototyping and incremental delivery to cope with change.

Processes Feature Matrix

Process	Distinct Phases	Incremental delivery	iterative	Evolving specification	Timeboxed iterations	Risk management
Waterfall	●					
Incremental	In parallel	●	no	no		
Evolutionary	In parallel	●	●	●		
Spiral	●	●	●	●		●
Unified Process	In parallel	●	●	●	●	
Generic Agile	In parallel	●	●	●		
Scrum	In parallel	●	●	●	●	●

Key points

- Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.