

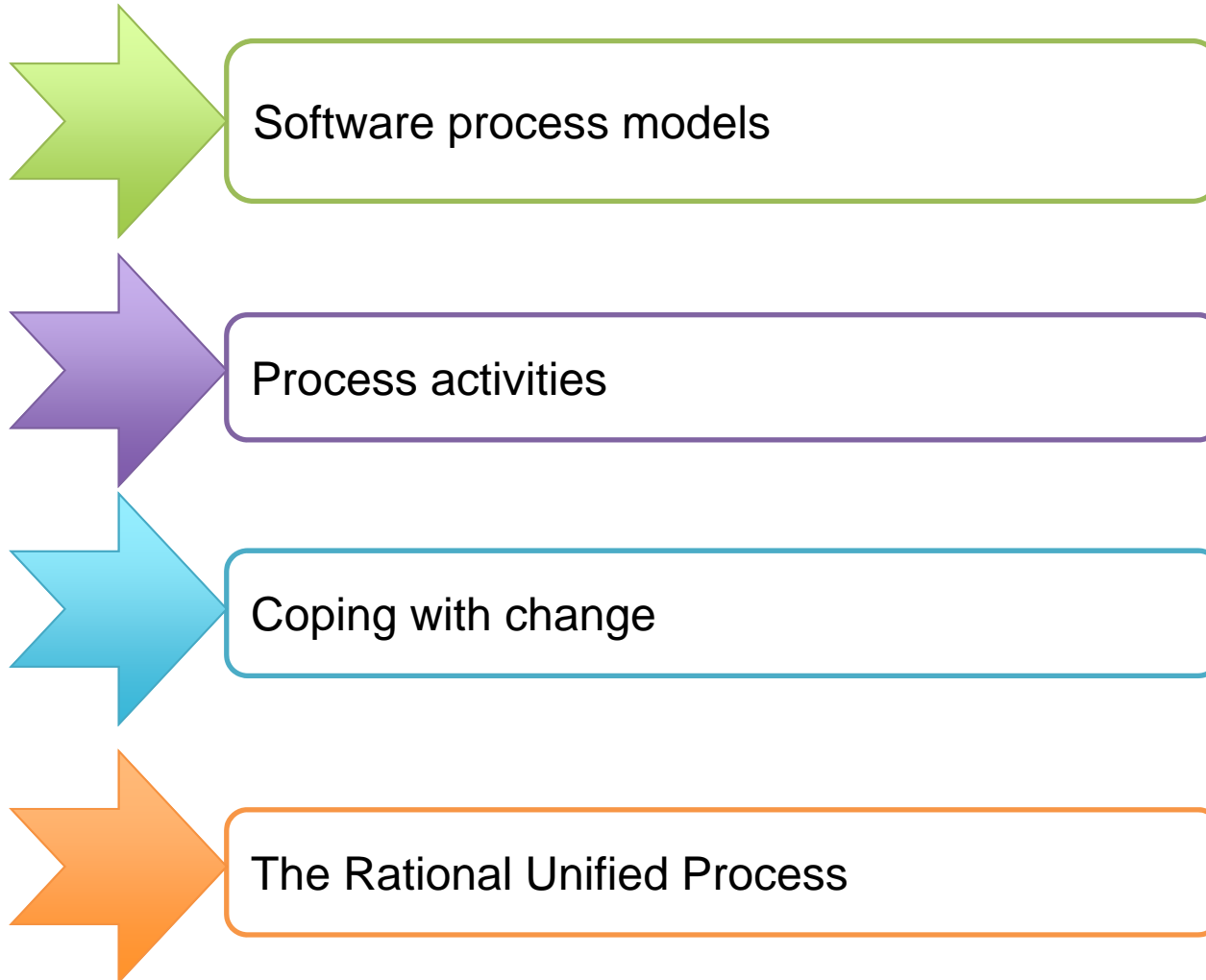


## Chapter 2 – Software Processes

### Lecture 1

# Topics covered

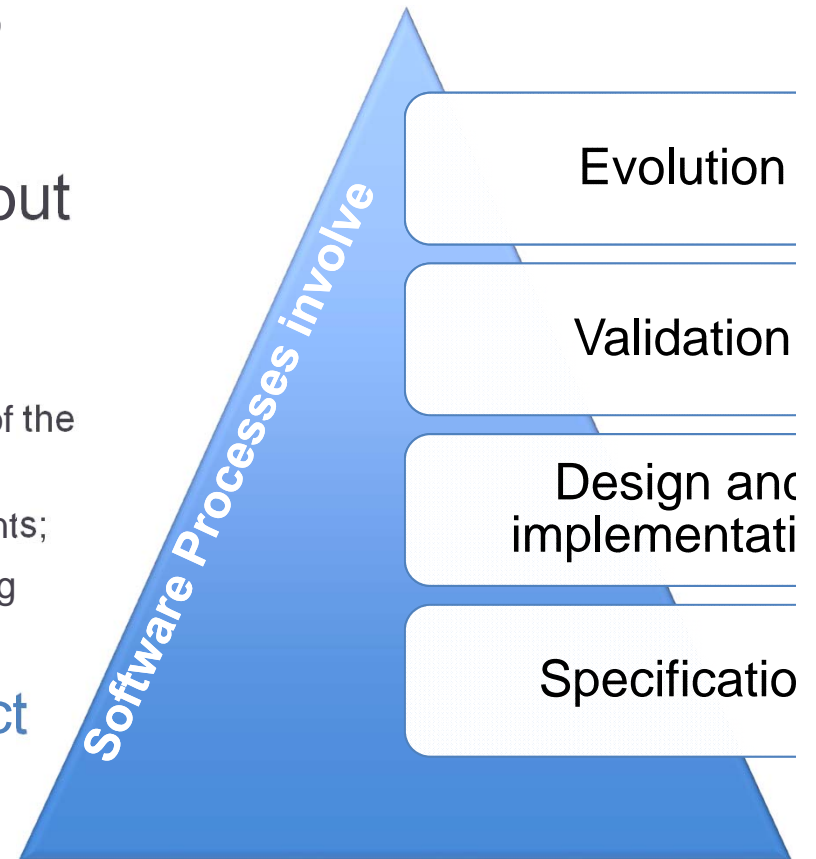
---



# The software process



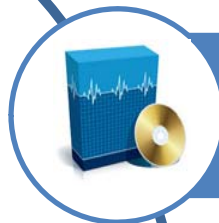
- A structured **set of activities** required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – 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.



# Software process descriptions



When we describe and discuss processes, we usually talk about the activities in these processes such as **specifying** a data model, **designing** a user interface, etc. and the **ordering** of these activities.



## Products

- The outcomes of a process activity;



## Roles

- Reflect the responsibilities of the people involved in the process;



## Pre- and post-conditions

- Statements that are true before and after a process activity has been enacted or a product produced.

# Plan-driven and agile processes

---



- Plan-driven 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.

# Software process models

---



## The waterfall model

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

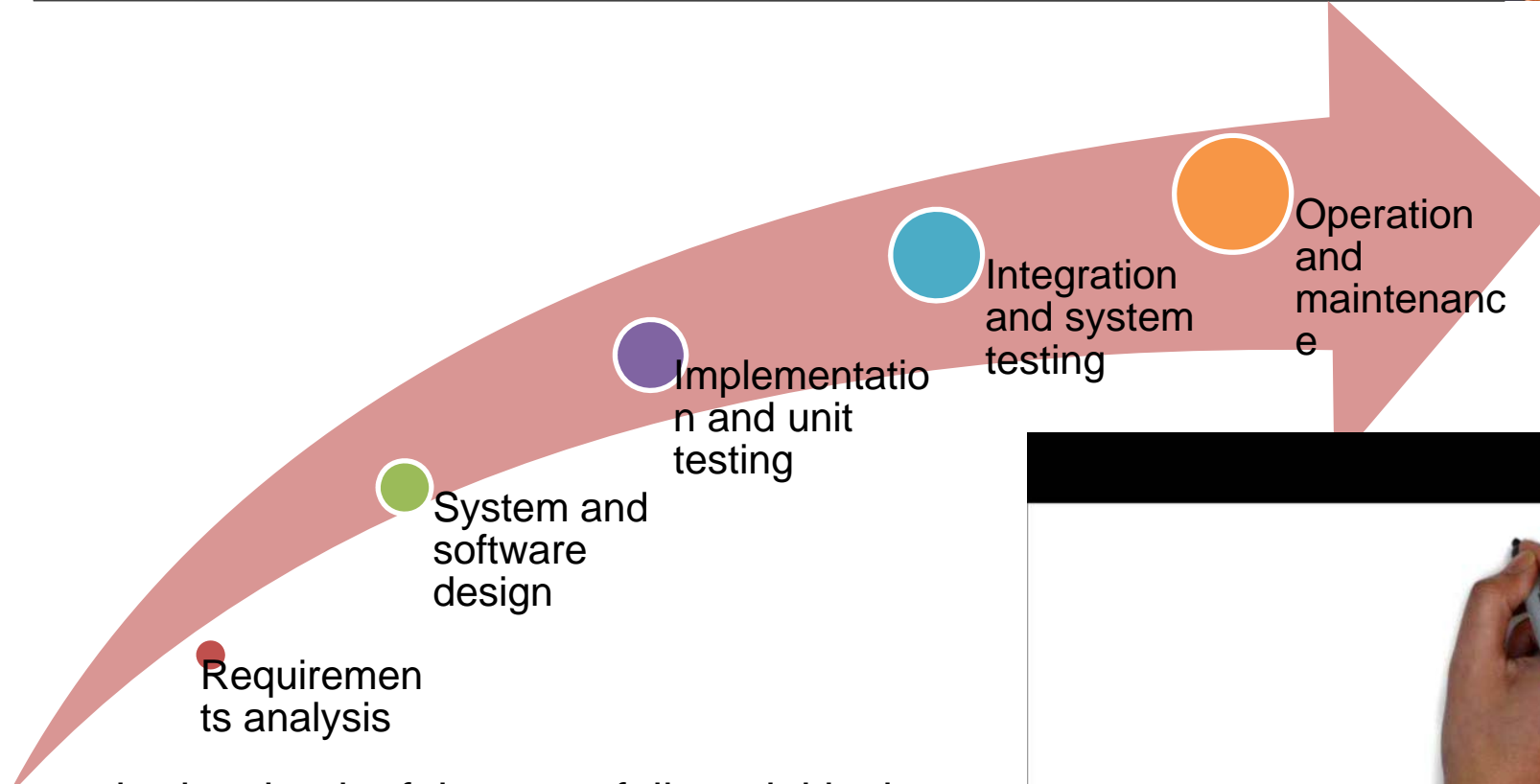
## Incremental development

- Specification, development and validation are interleaved.
- May be plan-driven or agile.

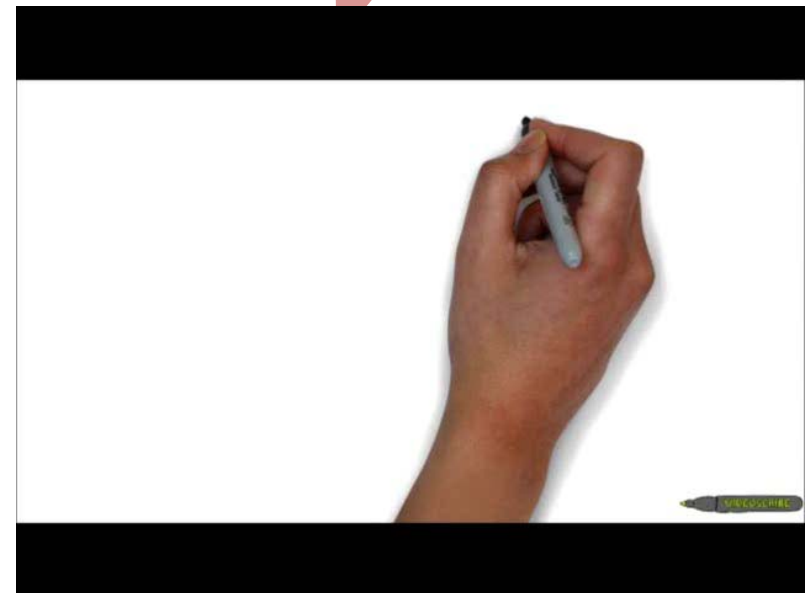
## Reuse-oriented software engineering

- The system is assembled from existing components.
- May be plan-driven or agile.

# Waterfall model phases



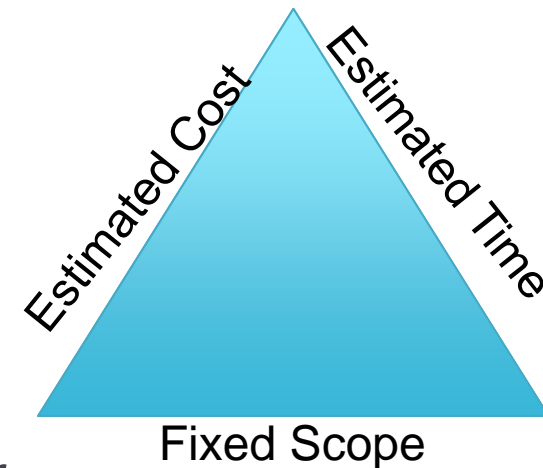
The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.



# Waterfall model problems

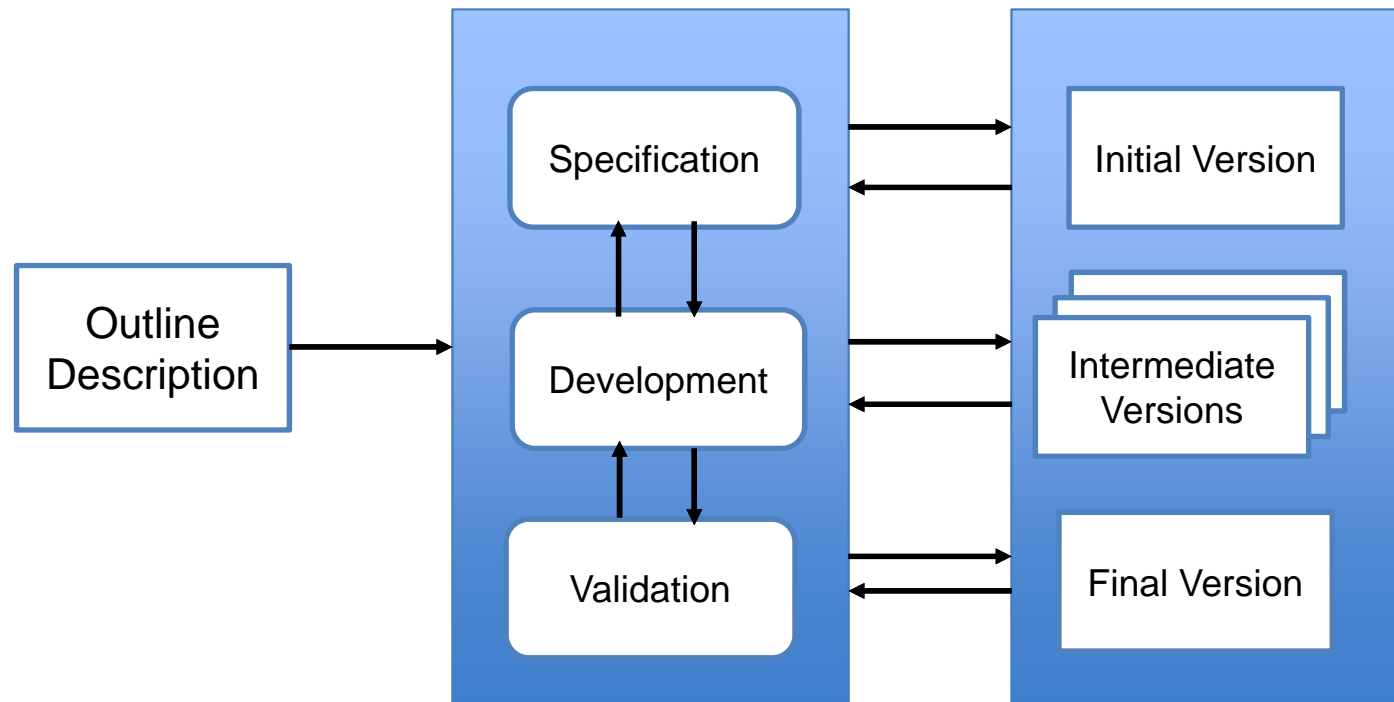
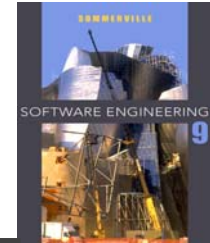


- Difficult to respond to **changing customer requirements**
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- Suitable large systems engineering projects where a system is **developed at several sites**.
- In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.





# Incremental development



# Incremental development benefits

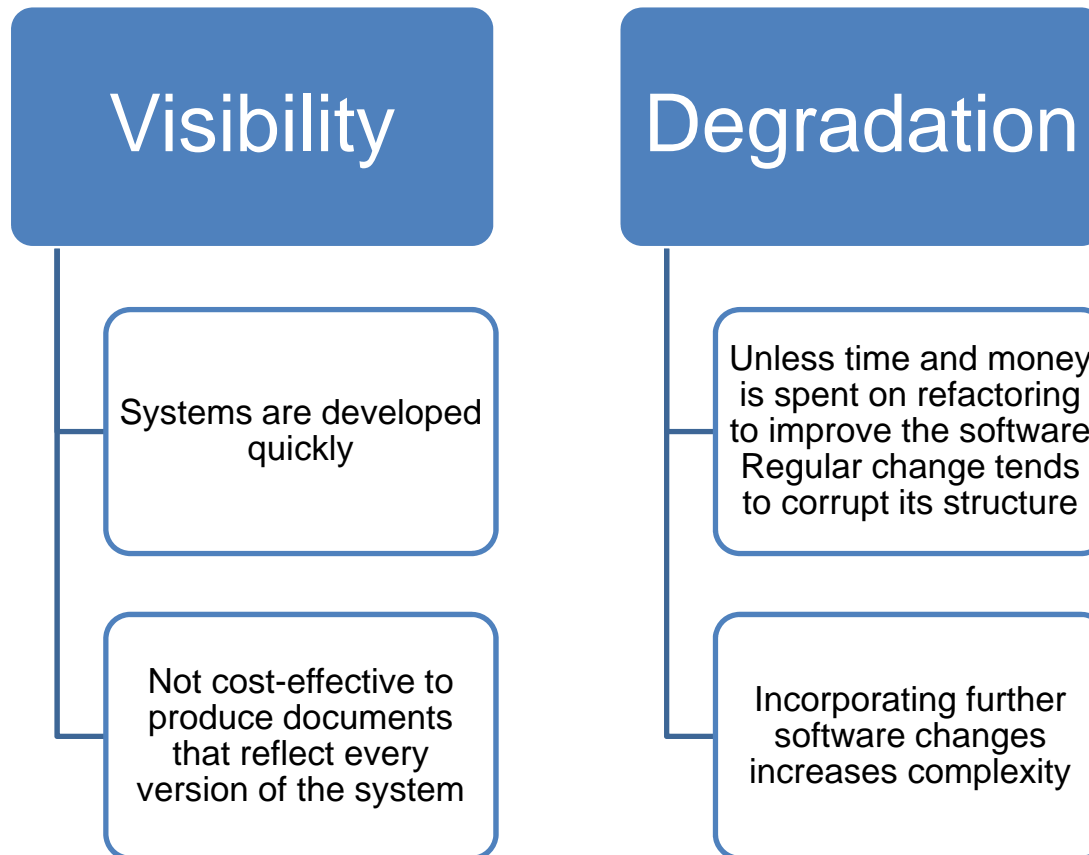


- The cost of **accommodating changing customer** requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- Customers are able to use and **gain value** from the software earlier than is possible with a waterfall process.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Concurrent Activities



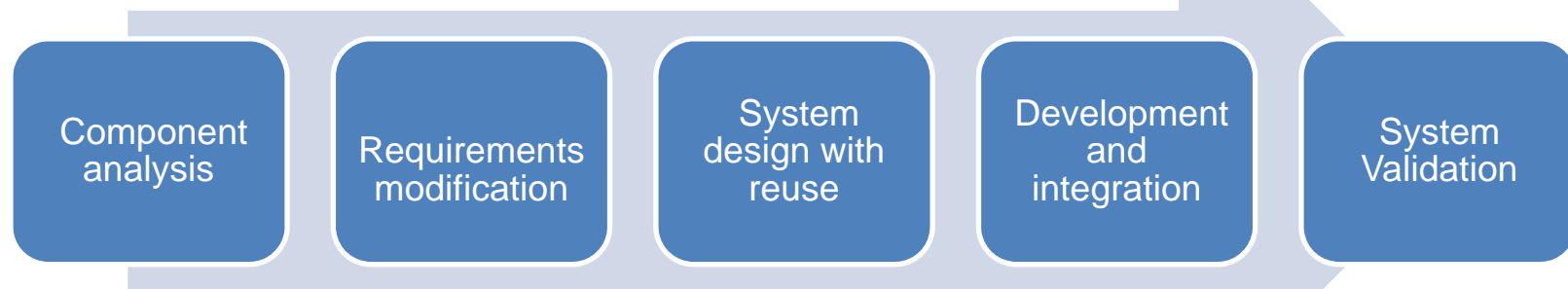
# Incremental development problems



# Reuse-oriented software engineering Stages



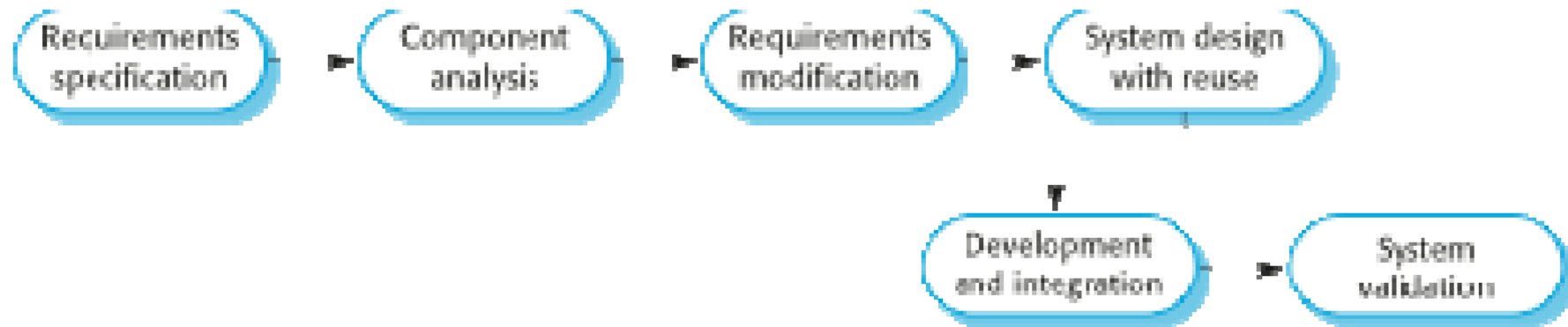
- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.



- Reuse is now the **standard approach** for building many types of business system



# Reuse-oriented software engineering



# Types of software component



Web services that are developed according to service standards and which are available for remote invocation.



Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.



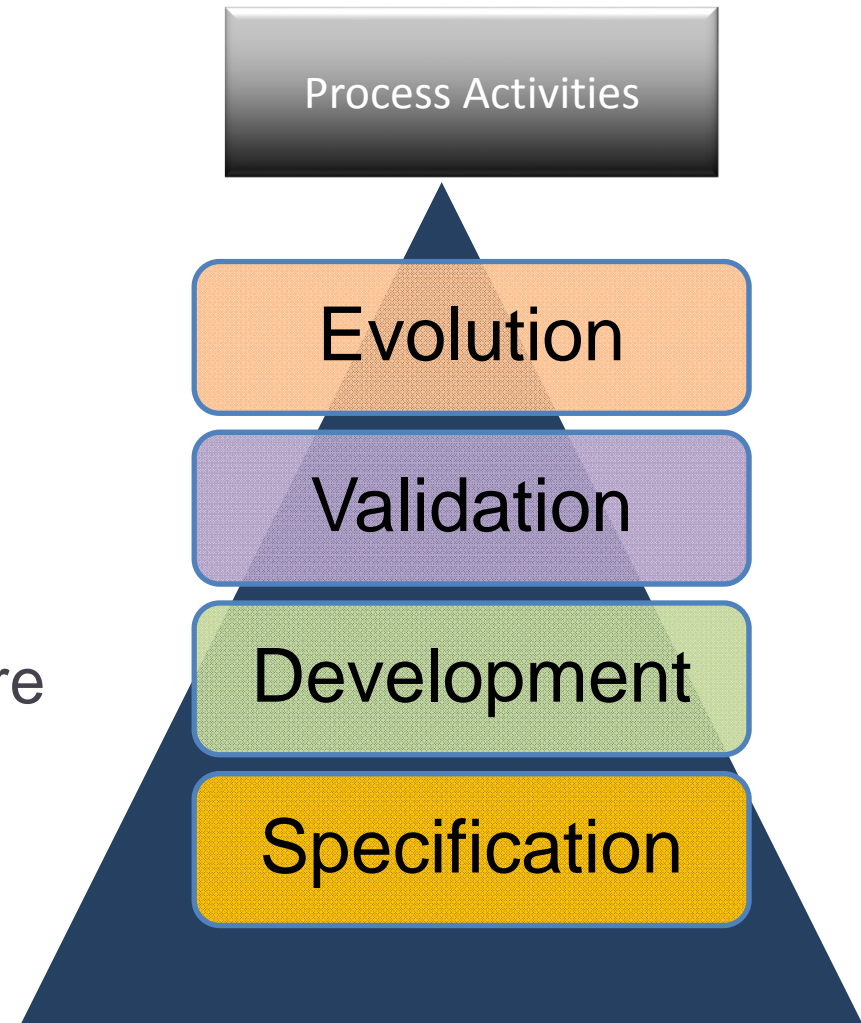
Stand-alone software systems (COTS) that are configured for use in a particular environment.



# Process activities



- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- In the waterfall model, they are organized in sequence, whereas in incremental development they are inter-leaved.



# Software specification

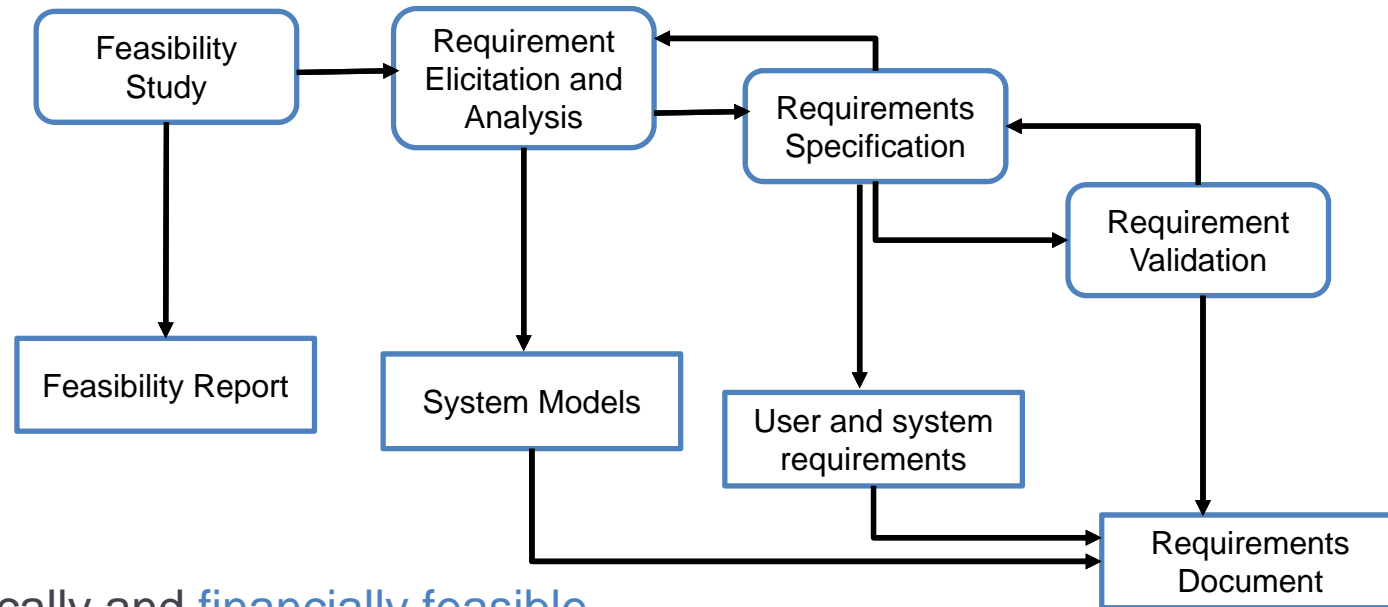
---



- The process of establishing what services are required and the constraints on the system's operation and development.
- Requirements engineering process
  - Feasibility study
    - Is it technically and financially feasible to build the system?
  - Requirements elicitation and analysis
    - What do the system stakeholders require or expect from the system?
  - Requirements specification
    - Defining the requirements in detail
  - Requirements validation
    - Checking the validity of the requirements

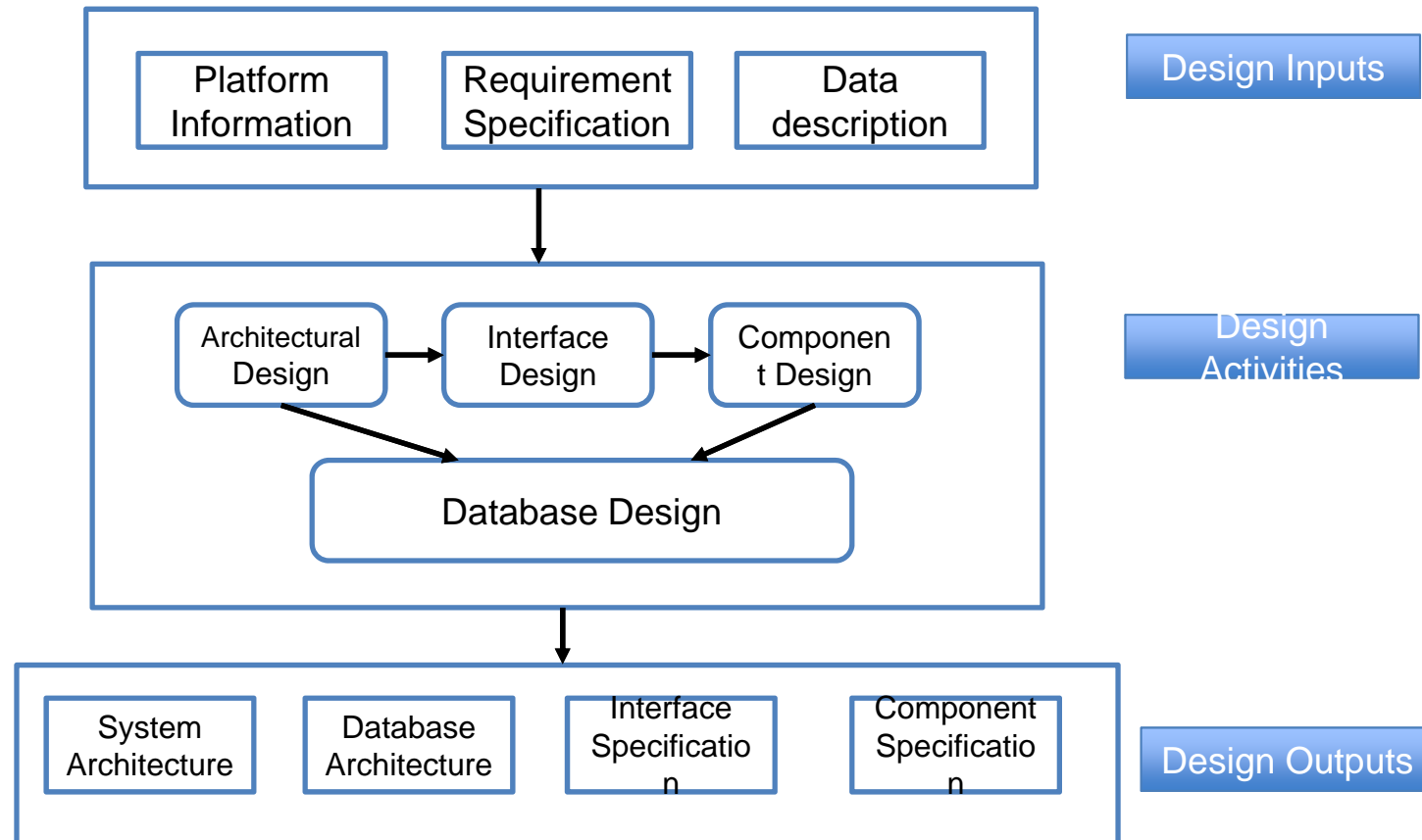


# The requirements engineering process

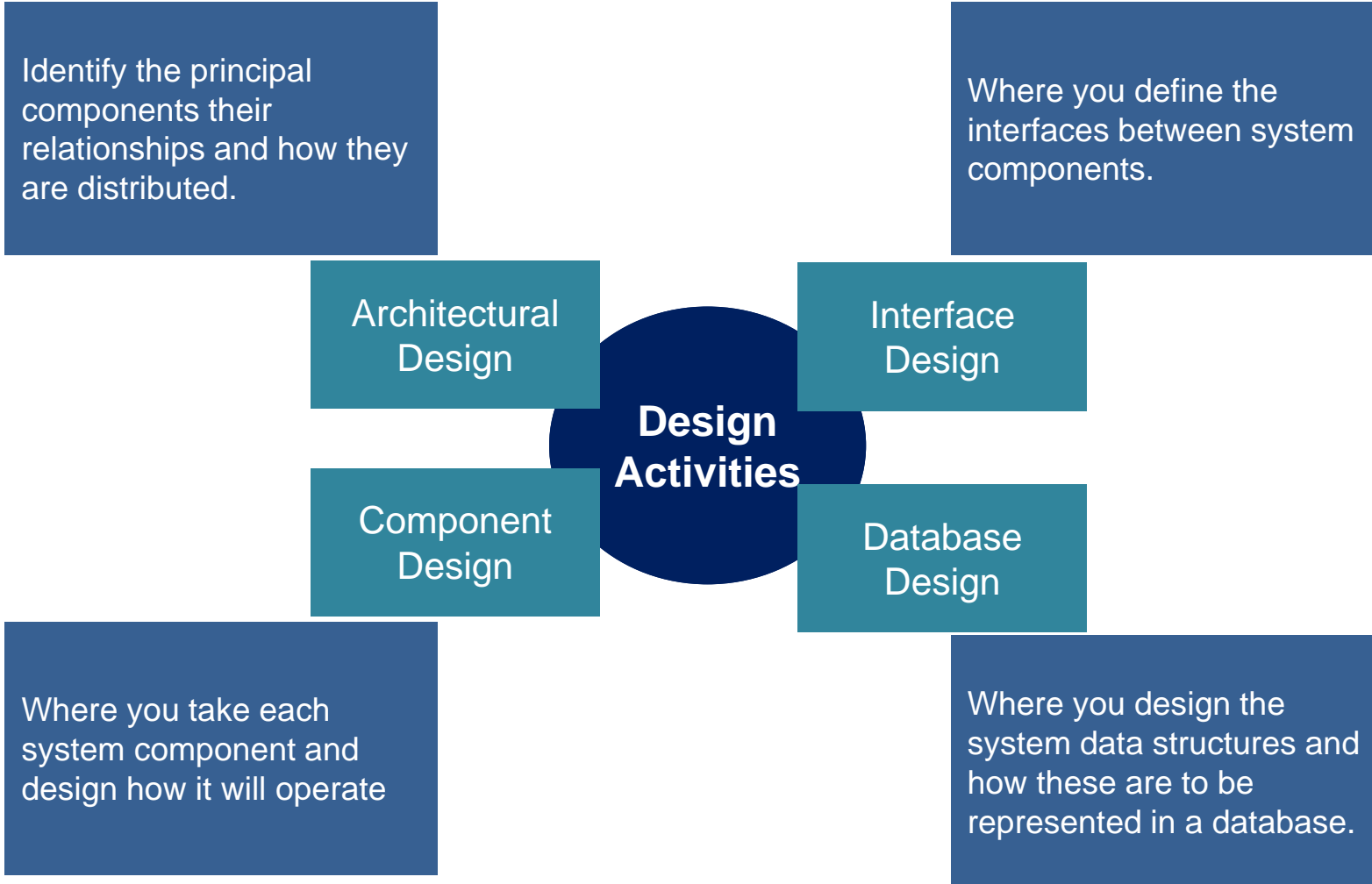


- Is it technically and **financially feasible** to build the system?
- What do the **system stakeholders require** or expect from the system?
- Defining the **requirements** in detail
- Checking the **validity** of the requirements

# A general model of the design process



# Design activities



# Software validation



Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.



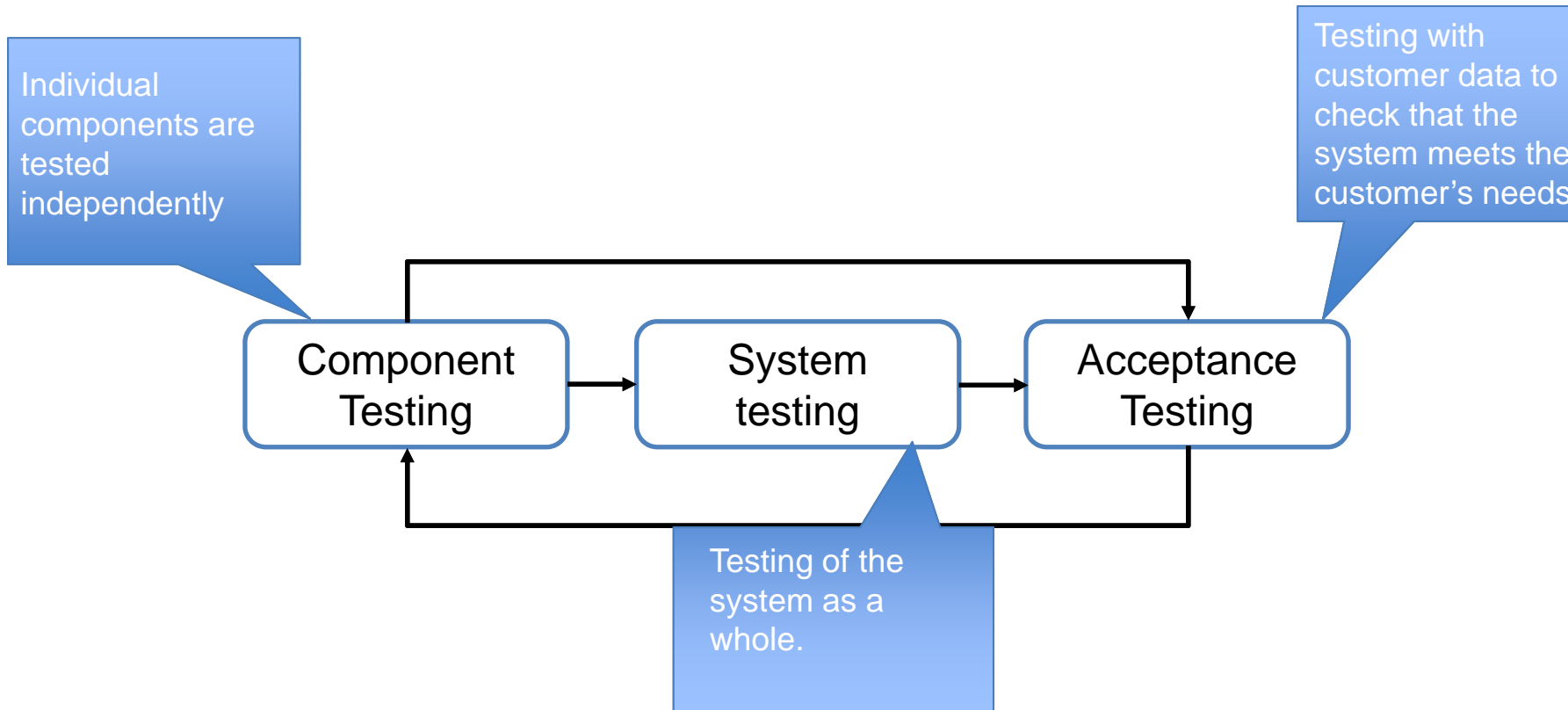
System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.



Test cases that are derived from the specification of the **real data** to be processed by the system.



# Stages of testing



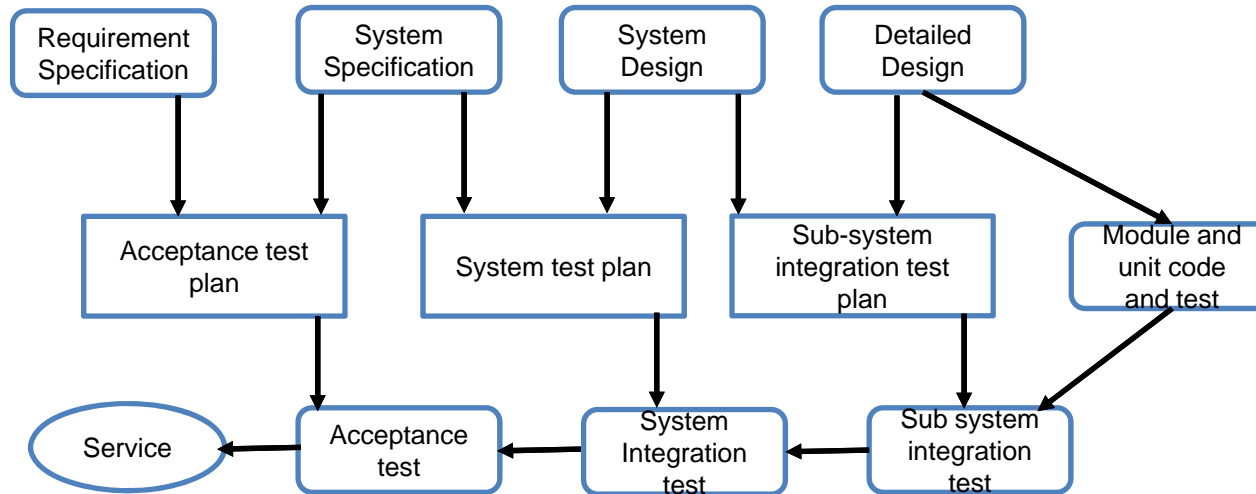
# Testing stages

---



- Development or component testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
- System testing
  - Testing of the system as a whole. Testing of emergent properties is particularly important.
- Acceptance testing
  - Testing with customer data to check that the system meets the customer's needs.

# Testing phases in a plan-driven software process



# Software evolution

---



- Software is **inherently flexible** and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

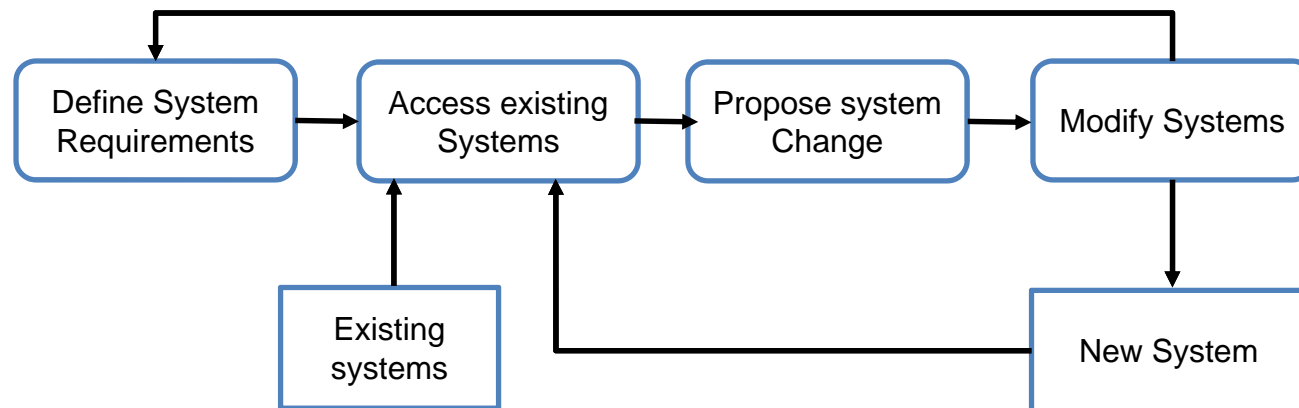




# System evolution



- To maintain the value of these software to the business, they must be **changed and updated**
- The majority of the software budget is devoted to changing and **evolving existing software** rather than developing new software.



# Key points

---



- Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes. Examples of these general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.

# Key points



- **Requirements engineering** is the process of developing a specification.
- **Design and implementation** process transforms a specification into an executable software system.
- **Software validation** is the process to check that the system conforms to its specification.
- **Software evolution** takes place when you change existing software systems to meet new requirements.



Illustration by Chris Gash



ClipartOf.com/231204



© Can Stock Photo - csp12956693



## Chapter 2 – Software Processes

### Lecture 2

# Coping with change



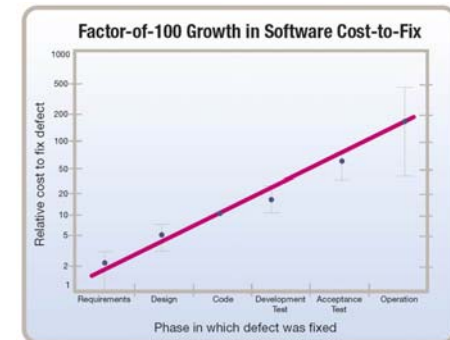
- Change is inevitable in all large software projects.
  - Business changes lead to new and changed system requirements
  - New technologies open up new possibilities for improving implementations
  - Changing platforms require **application changes**
- Change leads to rework so the **costs of change** include both rework as well as the costs of implementing new functionality



# Reducing the costs of rework



- Change avoidance, where the software process includes activities that can **anticipate possible changes** before significant rework is required.
- **Change tolerance**, where the process is designed so that changes can be accommodated at relatively low cost.
  - This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.



# Software prototyping

---

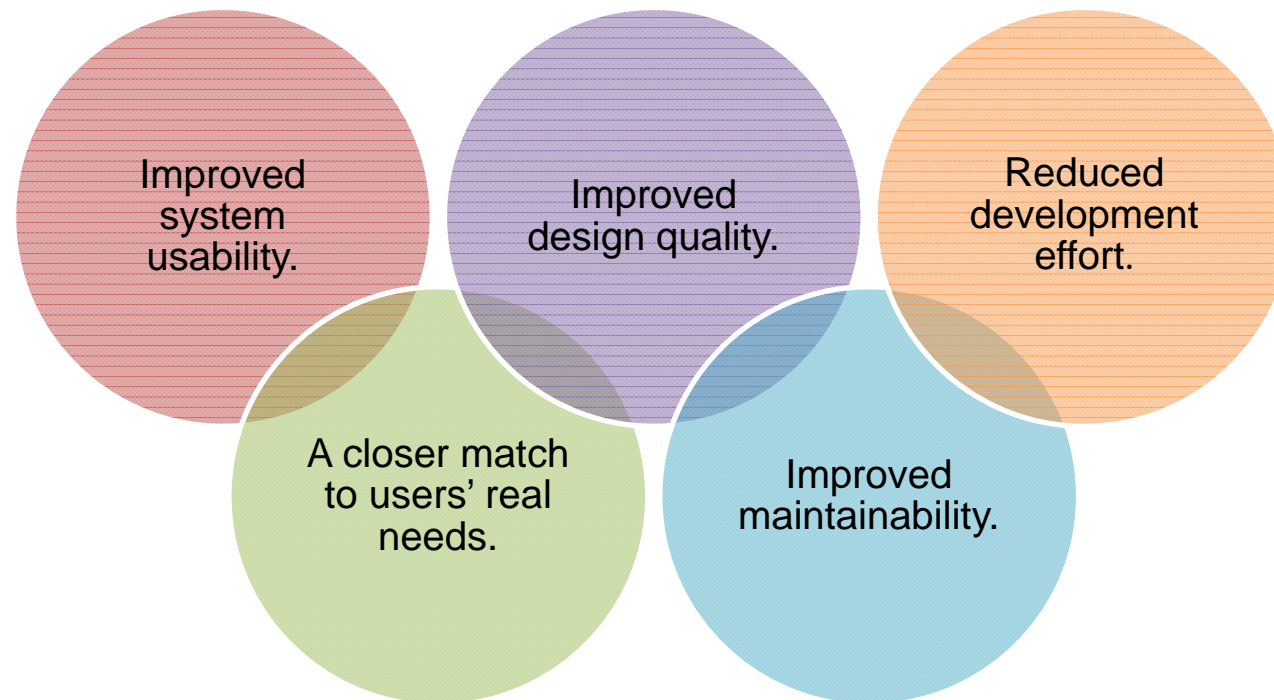


- A prototype is an **initial version** of a system used to demonstrate concepts and try out design options.
- A prototype can be used in:
  - The requirements engineering process to help with requirements **elicitation and validation**.
  - In design processes to explore options and develop a **UI design**.
  - In the testing process to run back-to-back tests.



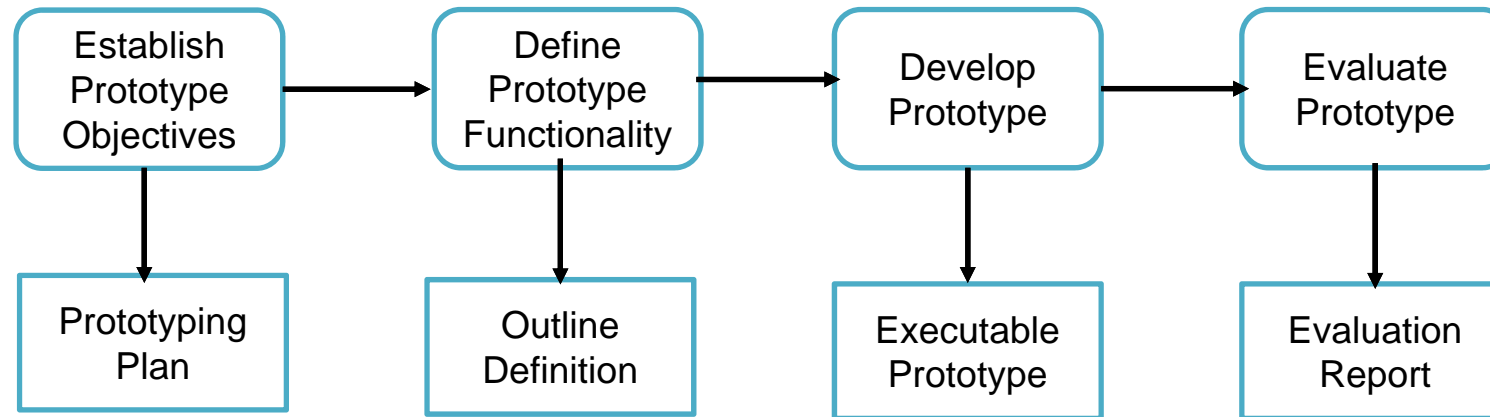
# Benefits of prototyping

---

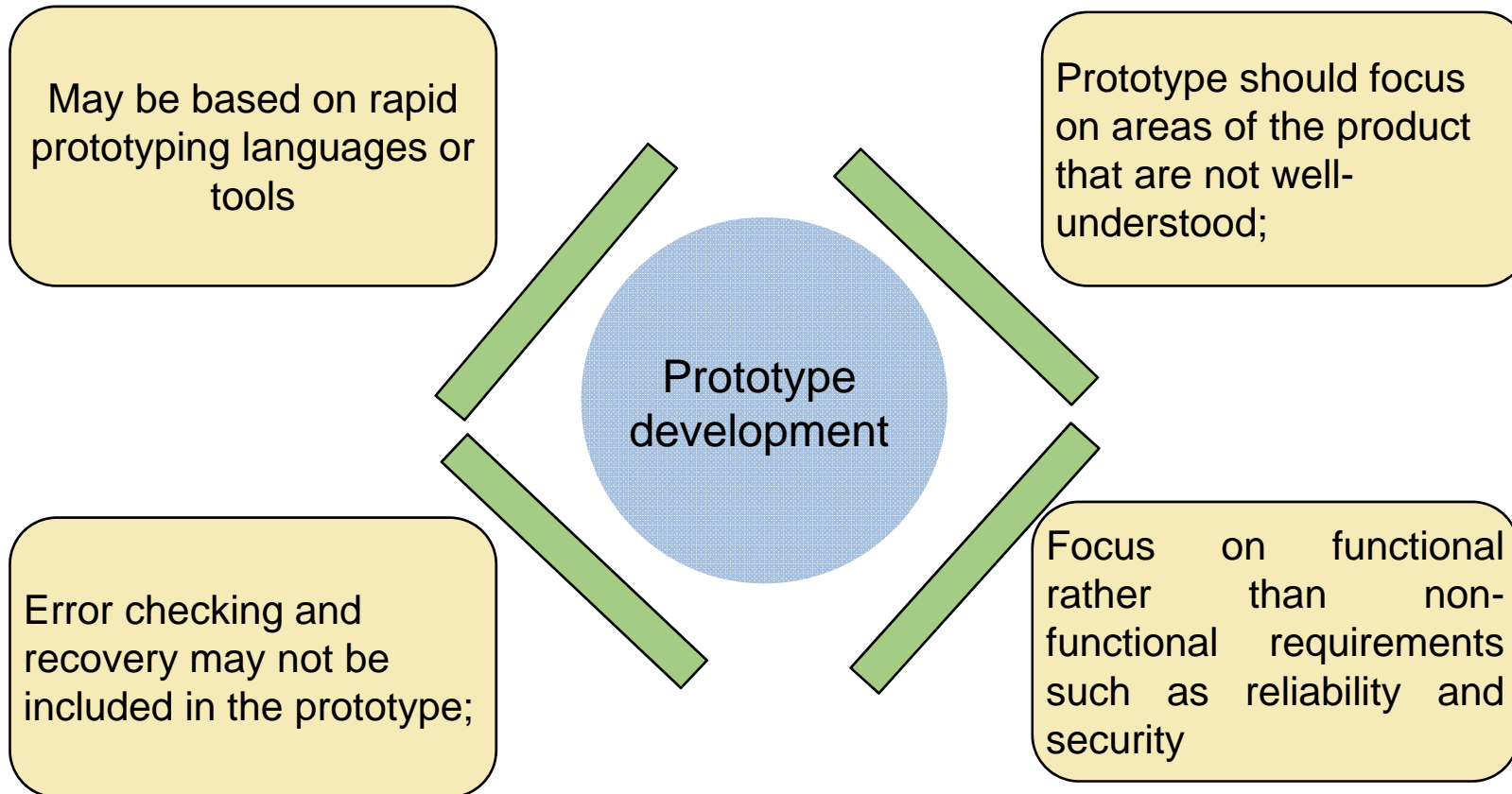




# The process of prototype development

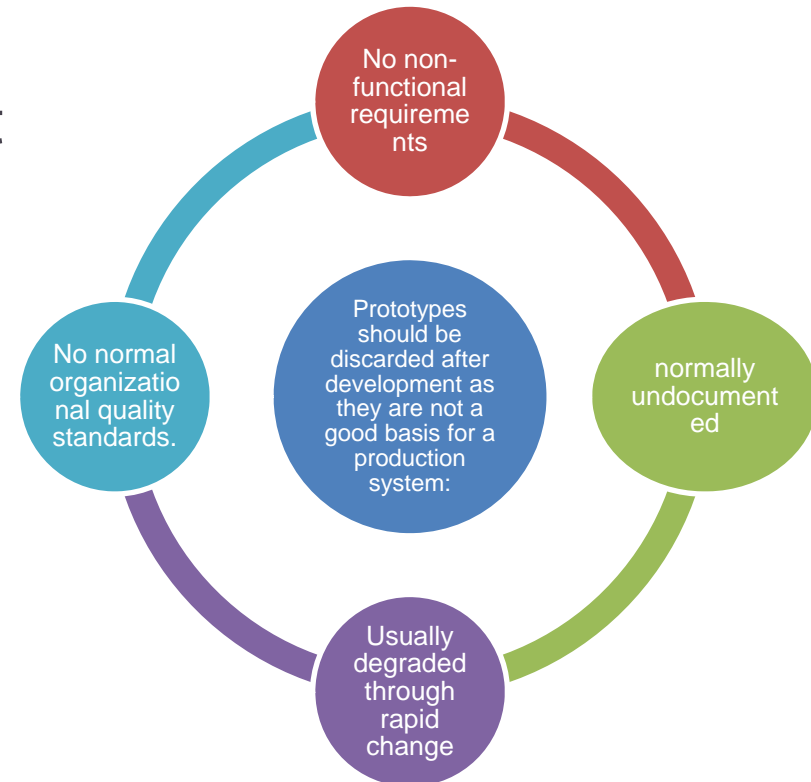


# Prototype development



# Throw-away prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - impossible to tune system to meet non-functional requirements;
  - Prototypes are normally undocumented;
  - The prototype structure is usually degraded through rapid change;
  - The prototype probably will not meet normal organizational quality standards.



# Incremental delivery

---



- 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 and delivery



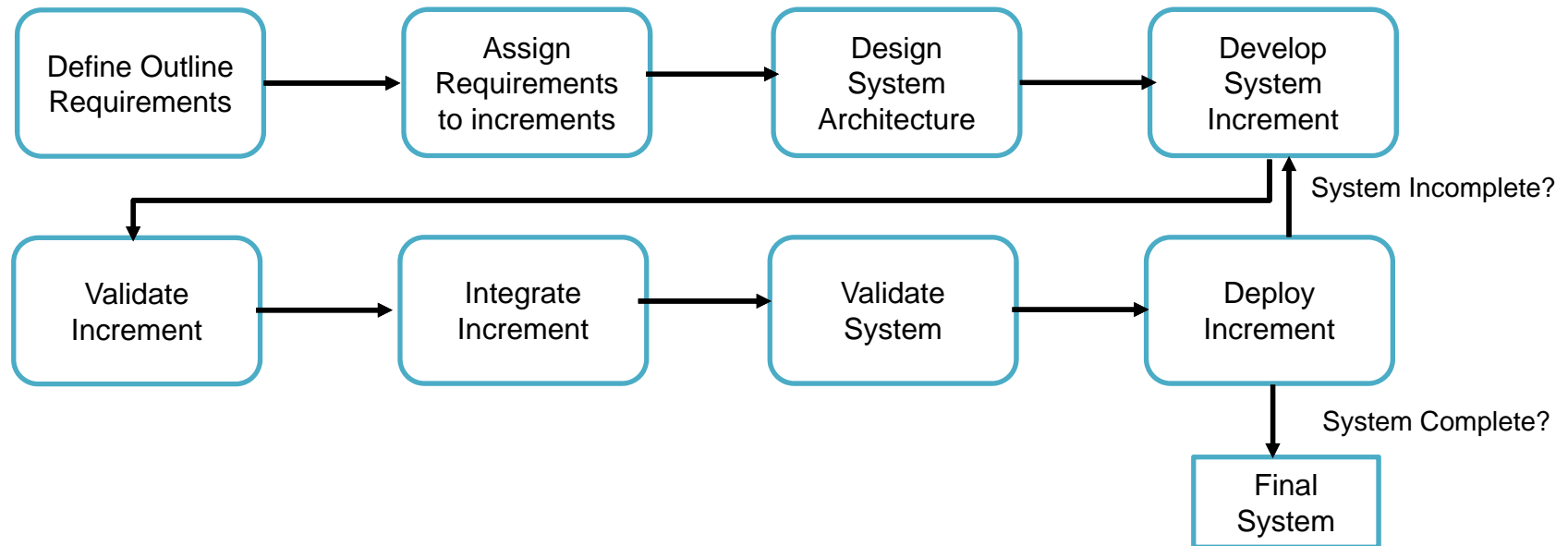
## Incremental development

- Develop the system in increments and **evaluate each increment** before proceeding to the development of the next increment
- Normal approach used in **agile methods**
- **Evaluation done** by user/customer proxy.

## Incremental delivery

- **Deploy** an increment for use by end-users
- More realistic evaluation about **practical use** of software
- Difficult to implement for replacement systems as increments **have less functionality** than the system being replaced.

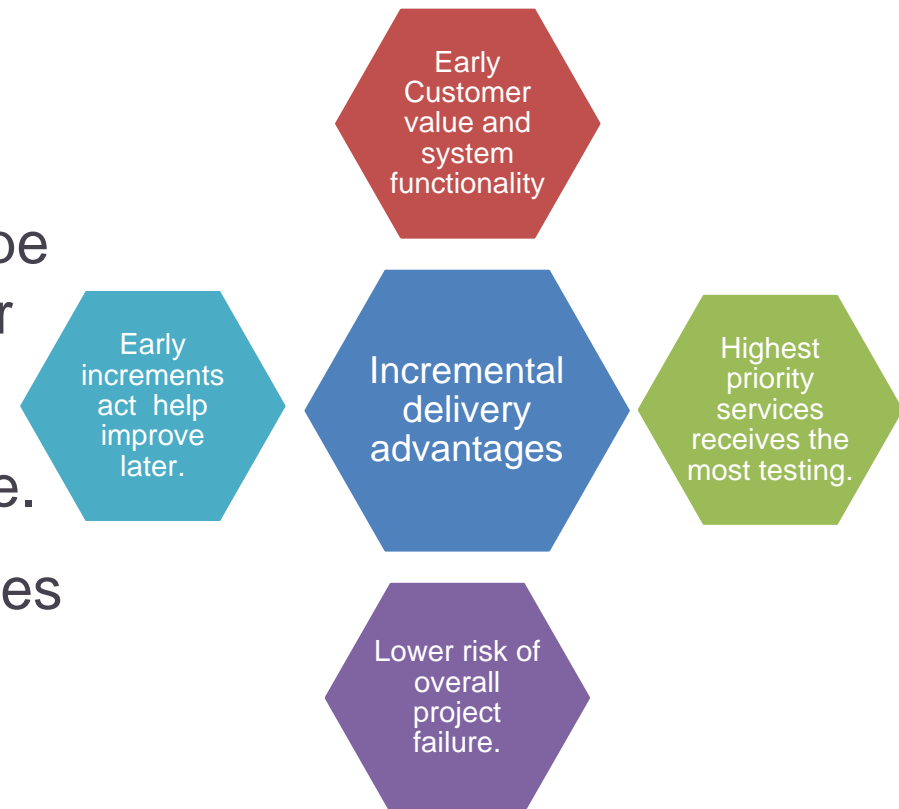
# Incremental delivery



# Incremental delivery 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 delivery problems

---



## Basic facilities

---

- Most systems require a set of basic facilities that are used by different parts of the system.
  - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.

## Parallel Development

---

- The essence of iterative processes is that the specification is developed in conjunction with the software.
  - However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.



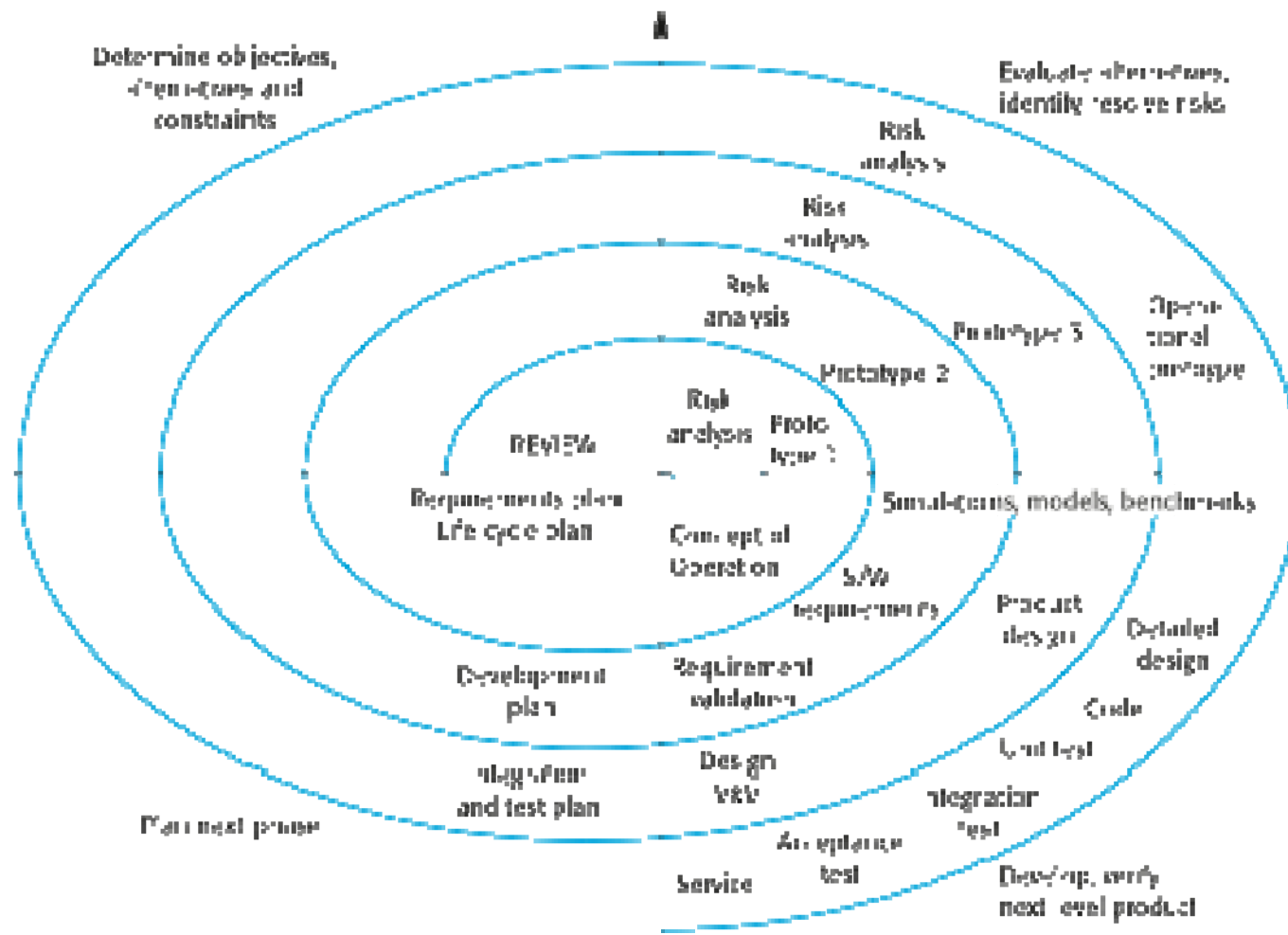
# Boehm's spiral model

---



- Process is represented 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.

# Boehm's spiral model of the software process



# 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.

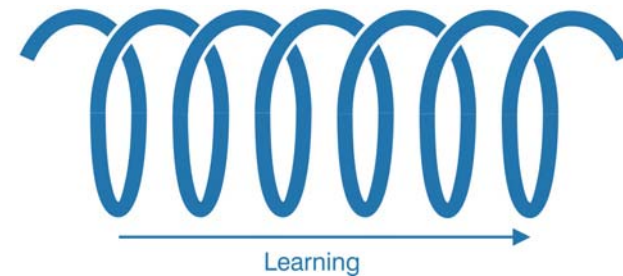
## Planning

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

# Spiral model usage



- Spiral model has been very influential in helping people think about iteration in software processes and introducing **the risk-driven approach** to development.
- In practice, however, **the model is rarely used** as published for practical software development.



# The Rational Unified Process

---



- A **modern generic process** derived from the work on the UML and associated process.
- Brings together aspects of the **3 generic process models**.
- Normally described from 3 perspectives
  - A **dynamic** perspective that shows phases over time;
  - A **static** perspective that shows process activities;
  - A **proactive** perspective that suggests good practice.

# Phases in the Rational Unified Process



Phase iteration



# RUP 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.

# RUP iteration

---



- In-phase iteration
  - Each phase is **iterative with results** developed incrementally.
- Cross-phase iteration
  - As shown by the loop in the RUP model, **the whole set of phases may be enacted incrementally.**



# Static workflows in the Rational Unified Process



<b>Workflow</b>	<b>Description</b>
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.

# Static workflows in the Rational Unified Process



<b>Workflow</b>	<b>Description</b>
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
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

# RUP good practice

---



- Develop software iteratively
  - Plan increments based on customer priorities and **deliver highest priority increments first.**
- Manage requirements
  - Explicitly **document customer requirements** and keep track of changes to these requirements.
- Use component-based architectures
  - Organize the system architecture as a set of **reusable components.**

# RUP good practice

---



- Visually model software
  - Use graphical UML models to present static and dynamic views of the software.
- Verify software quality
  - Ensure that the software meet's organizational quality standards.
- Control changes to software
  - Manage software changes using a change management system and configuration management tools.

# Key points

---



Processes should include activities to cope with change. This may involve a **prototyping phase** that helps avoid poor decisions on requirements and design.

---

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

---

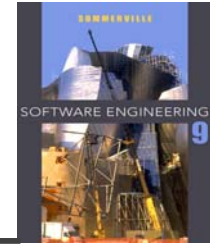
The Rational Unified Process is a **modern generic process model** that is organized into phases but separates activities from these phases.

# Useful Links

---

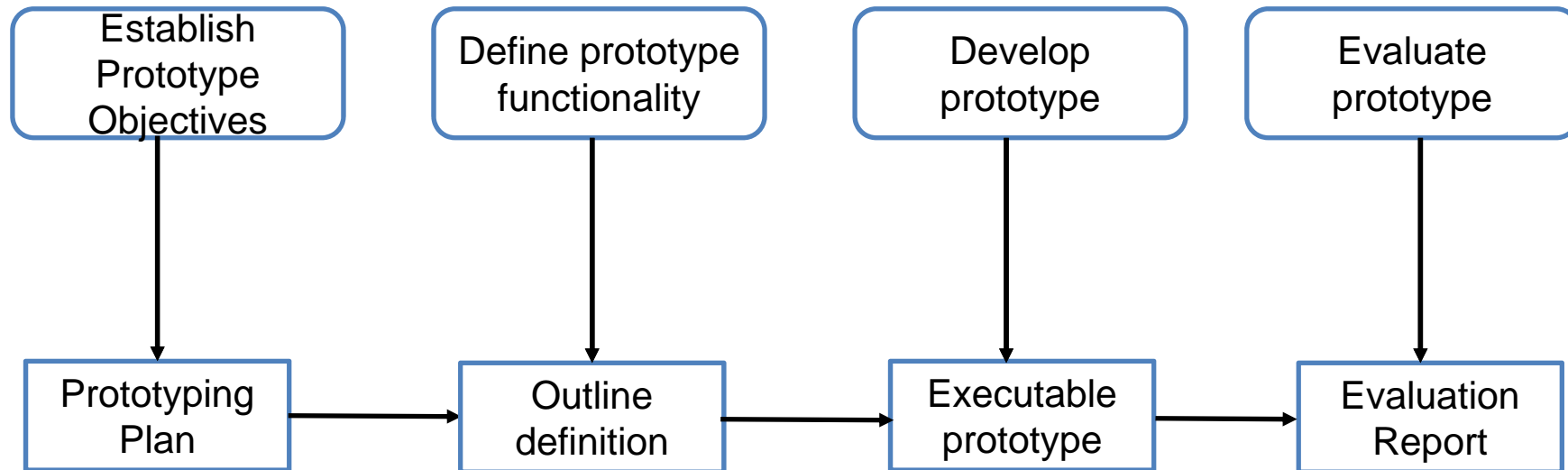


- [http://www.tutorialspoint.com/sdlc/sdlc\\_waterfall\\_model.htm](http://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm)
- <http://www.seguetech.com/blog/2013/07/05/waterfall-vs-agile-right-development-methodology>
- <https://axiosdev.wordpress.com/2012/11/26/agile-or-incremental-software-development-which-is-better/>
- [http://www.tutorialspoint.com/sdlc/sdlc\\_software\\_prototyping.htm](http://www.tutorialspoint.com/sdlc/sdlc_software_prototyping.htm)
- [http://www.tutorialspoint.com/sdlc/sdlc\\_spiral\\_model.htm](http://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm)
- <http://www.agilemodeling.com/essays/agileModelingRUP.htm>
- <http://www.techterms.com/definition/rup>



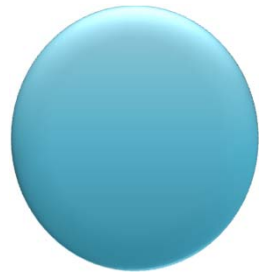
# Thank You

# The process of prototype development





# Benefits of prototyping



- Improved System Usability
- Improved design quality.
- Improved maintainability.
- A closer match to users' real needs
- Reduced development effort.