Software Engineering I

Chapter 2

Software Processes

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. It presents a description of a process from some particular perspective.

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

Software process models

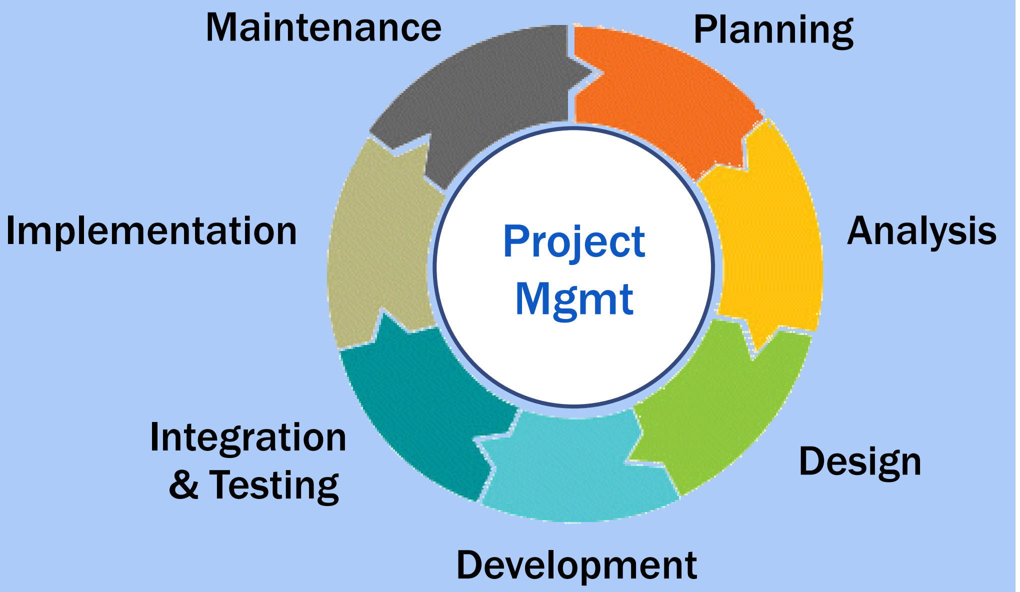
- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- Integration and configuration
  - The system is assembled from existing configurable components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

System Development Life Cycle

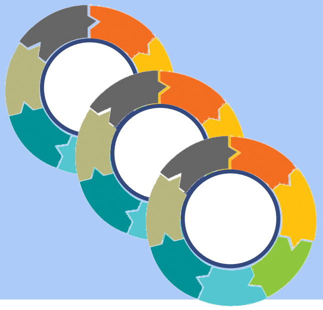
- No matter whether a project is plan-driven or agile, there are a series of activities that must be undertaken for every software engineering project
- A systems development life cycle (SDLC) is composed of a number of clearly defined and distinct work phases which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems.
- There is no standard SDLC model, as each institution typically builds their own phases – however the tasks performed in each phase remain relatively constant from model to model

A representative SDLC

- This SDLC has seven phases
  - Planning
  - Analysis
  - Design
  - Development
  - Integration and Testing
  - Implementation
  - Maintenance



Agile methodologies go through all these phases multiple times

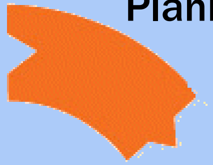


Project Management processes run in parallel to system development processes and will be discussed in a later lecture.

Planning

- The *planning phase* of the SDLC involves determining a solid plan for developing your information system.
- A *project manager,* an individual who is an expert in project planning and management may or may not be involved at this stage.
- Three primary activities:
  - Define the system to be developed.
    - Identify and select the system for development or determine which system is required to support the strategic goals of your organization.
    - *Critical success factors (CSF)* are factors critical to your organization's success.
  - Set the project scope.
    - The *project scope* clearly defines the ***high-level*** system requirements and is the most basic definition of the system and is usually no longer than a paragraph.
  - Define the ***high level*** project plan.
    - At this stage of the project, the project plan may consist of nothing more than p*roject milestones* represent key dates by which you need a certain group of activities performed.

A full blown project plan defines the what, when, and who questions of systems development activities including all activities, the individuals, or resources, who will perform the activities, and the time required to complete each activity

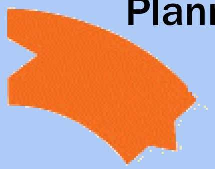


Begins with an idea

Ends with organizational approval

Planning: The Project Initiation Document (PID)

- What the project is aiming and planning to achieve
- The reason for the importance of meeting these aims.
- Approved by Project Board (all stakeholders and mgmt)
- Not regularly updated post-approval during project stages
- May contain all or most of the sections below



- **Project Scope Statement**
  - What is in scope and out of scope
  - Will project be delivered in phases?
- **Project Background**
  - Why and how project was created.
  - History and context
- **Feasibility Analysis**
- **Assumptions, Dependencies and Constraints**
- **Initial Project Plan**
  - High level milestones and dates
- **Organization and Governance**
  - Team Organization Chart.
  - Governance structure and checkpoints
- **Communication Plan**
  - e.g. weekly project team meetings, biweekly governance meetings, minutes…
- **Quality Plan**
  - Often a list of deliverables
- **Risk Assessment**
  - What are expected risks & mitigation strategies
- **Estimated Cost and ROI**

What makes for a good PID?

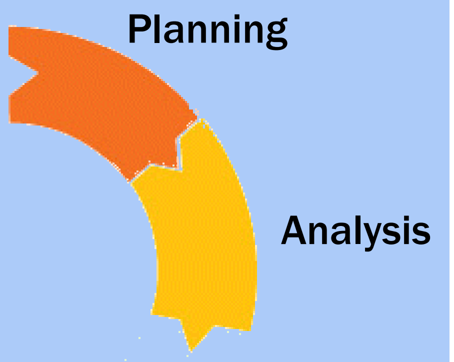
1. As an IT approver of this document, I am looking to understand whether I should allocate scarce IT resources for this project or choose another one.
   The "Go / No-Go" decision
   - Do I believe the author understands enough (at this early stage) about the technology that could be in play?
     Is this project feasible?
   - Do I feel the author has adequately defined the resources needed to complete this project?
     (Is it a team of 9? A team of 12? What roles are required? Does the team need a database professional? A UI specialist?)
   - Things will go wrong. Do I believe that the author understands the most important project risks? Does the author have strategies for risk mitigation?
   - Do I understand the business value of this project?
   - Has the author developed an approach / methodology to complete the project?
     (Is there a high-level plan that makes sense? What training will be required? How will transparency to stakeholders be achieved? Are there review points? Agile? Waterfall? Hybrid?

2. As a business approver of this document, I am looking to be satisfied that the PID adequately documents my needs
   - Does the project description cover the key aspects from my perspective?
   - Are my high-level requirements well stated?
   - Is the scope correct?
     (Are the right elements included Do the exclusions make sense?)

3. As a consumer of this document, I am looking to gain a good understanding of the project
   - What is the purpose of the project?
   - Why is the project important?
   - How will communications occur?

Analysis

- The analysis phase of the SDLC involves end users and IT specialists working together to gather, understand, and document the business requirements for the proposed system.
- The primary activity of this phase is to gather business requirements. Business requirements are the detailed set of knowledge worker requests that the system must meet to be successful.
- Methodologies treat this phase differently
  - Lock down requirements and only change if absolutely necessary
  - Define all requirements but allow/encourage revisions as project progresses
  - Allow requirements to be agilely added, removed and evolved over the course of the project
- Documentation standards vary with the methodology and the institution

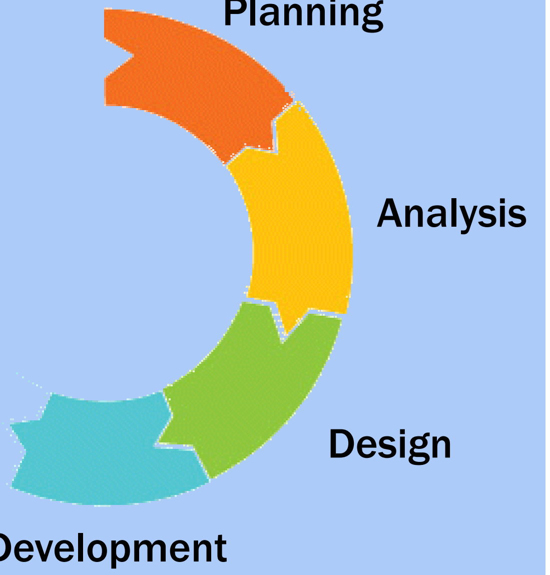


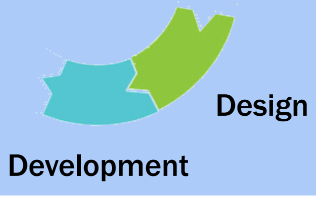
Begins with organizational approval of project

Ends?

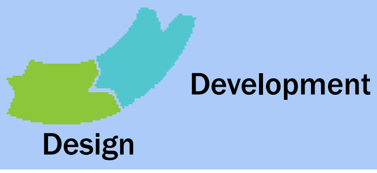
Design and Development

- The primary goal of the *design phase* is to build a technical blueprint of how the proposed system will work. Your point of view changes from a business perspective to a technical or physical perspective.
- The *technical architecture* defines the hardware, software, and required networking configuration to run the system.
- Defines and specifies the interfaces, parameters, and protocols used by product architecture and system architecture layers.
- Design documentation could act as
  - Specifications for development OR
  - An outcome of development *(or both!)*





OR



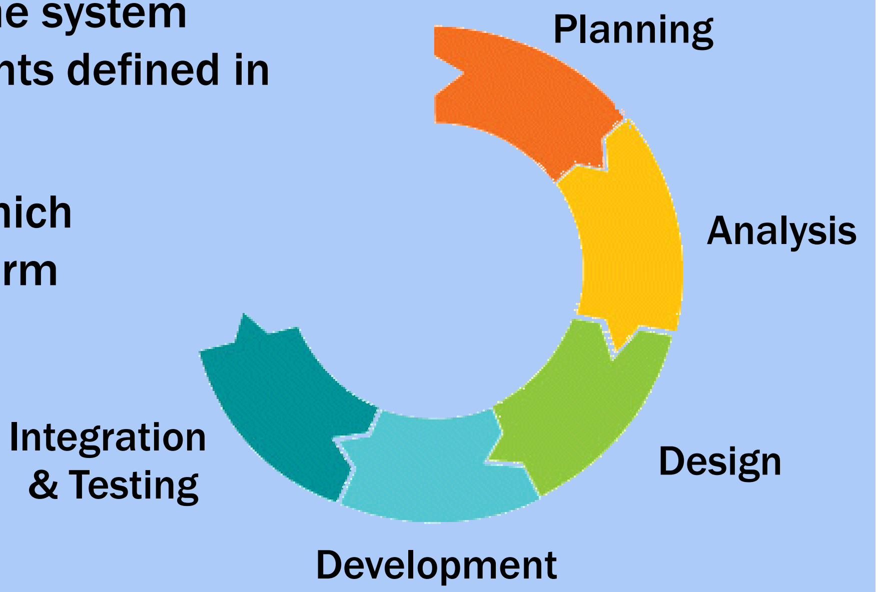
Begins after requirements are established.

- Technology should not constrain business needs
- Technical capability should inspire the business, but not cause scope creep

Ends?

Integration and Testing

- The testing phase of the SDLC verifies that the system works and meets all the business requirements defined in the analysis phase.
- First, you develop detailed test conditions, which are the detailed steps the system must perform along with the expected results of each step.
- Secondly, you actually perform the test.
- Types of testing
  - Ad hoc testing: normal testing performed by developer during coding process
  - Unit testing: test only a single "unit" of the code (say a module or a class), to see if it behaves as expected in isolation
  - Integration testing: individual units are combined and tested as a group
  - System testing: test the system as a whole.

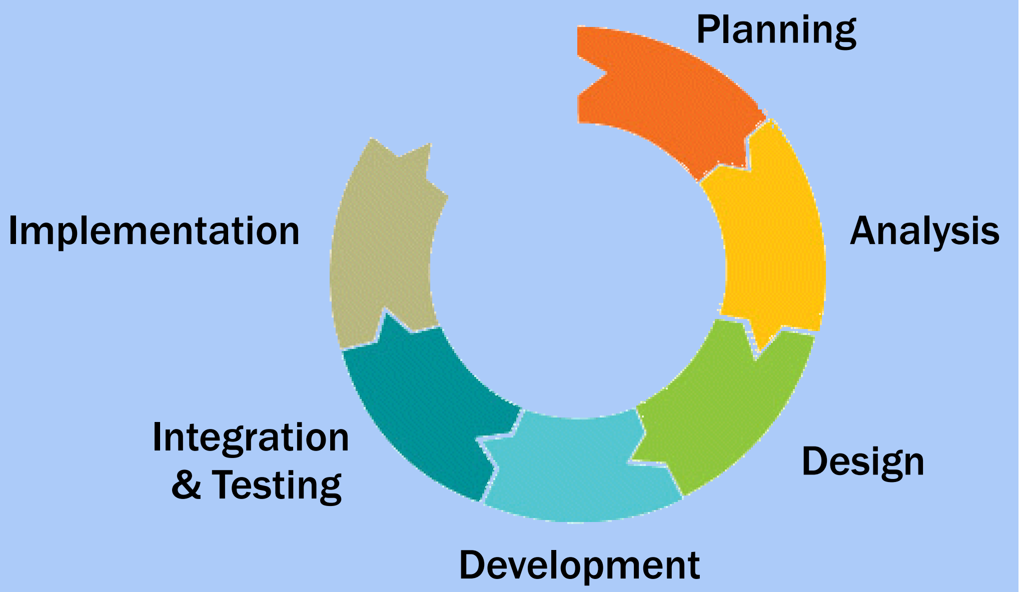


Begins once code and test plans are completed.

Ends when all testing is finished often with a User Acceptance document

Implementation

- During the implementation phase of the SDLC, you distribute the system to all the knowledge workers and they begin using the system to perform their everyday jobs
- Typical activities
  - Create Implementation Plan
  - Deploy system to a staging environment
  - Provide user documentation to the knowledge workers which explains how to use the system
  - Prepare the system support teams
  - Train knowledge workers

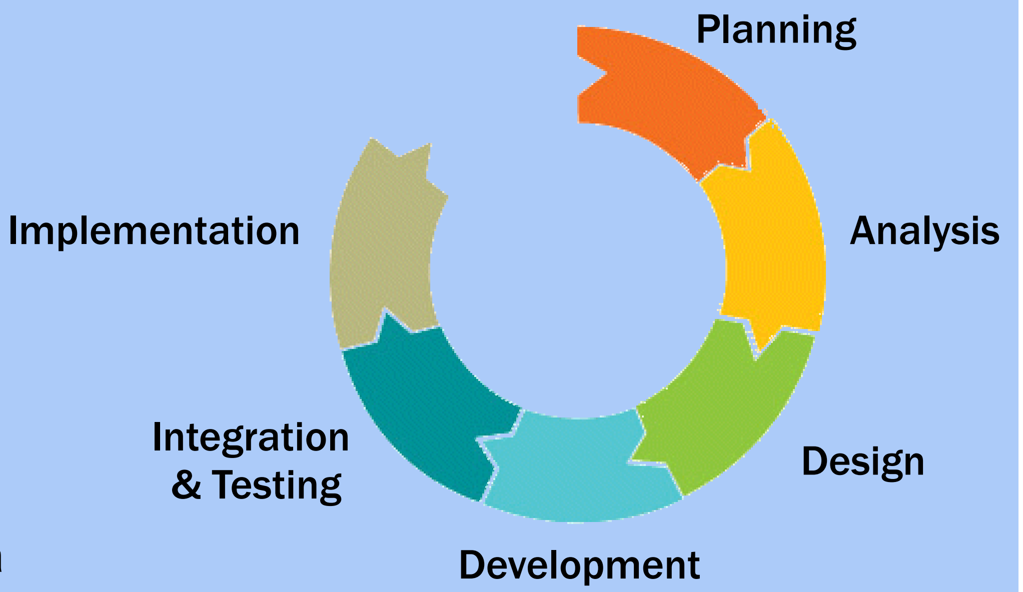


Begins at the conclusion of system testing

Ends when system is put into production

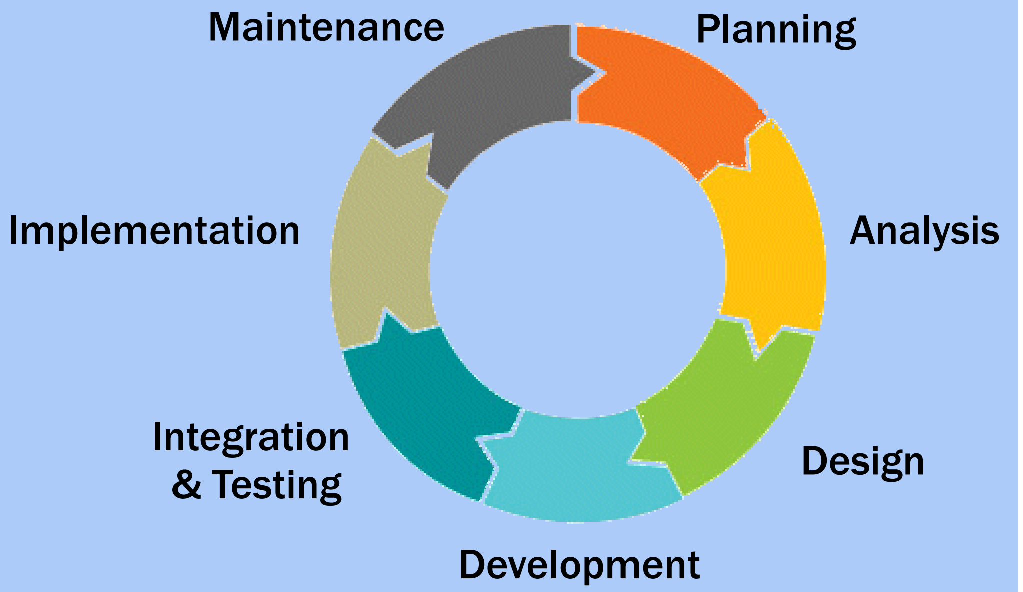
Implementation Techniques

- **Parallel implementation** – using both the old and new system until you are sure that the new system performs correctly.
- **Plunge implementation** – discarding the old system completely and using the new system
- **Pilot implementation** – having only a small group of people use the new system until you know it works correctly, and then adding the remaining people to the system.
- **Phased implementation** – implementing the new system one part at a time.



Maintenance

- Monitor and support the new system to ensure it continues to meet the business goals
- Change the system as your business changes
  - Minor changes and bug fixes can be processed through change management processes which also go through their own SDLC
  - Major enhancements often require a new Project Initiation Document and, if approved, spawn their own SDLC



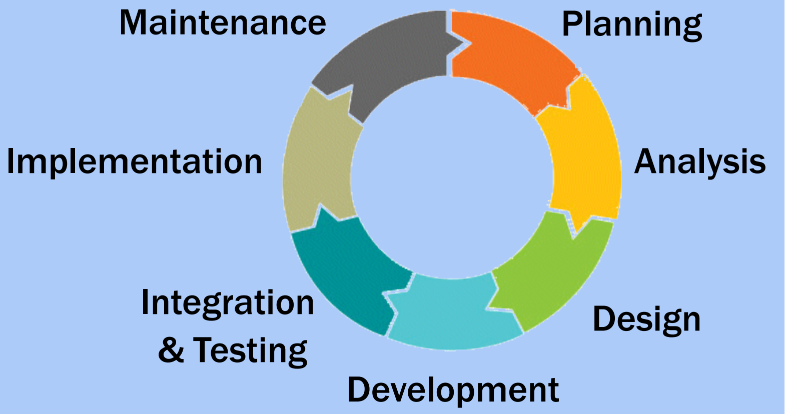
Begins when system is put into production

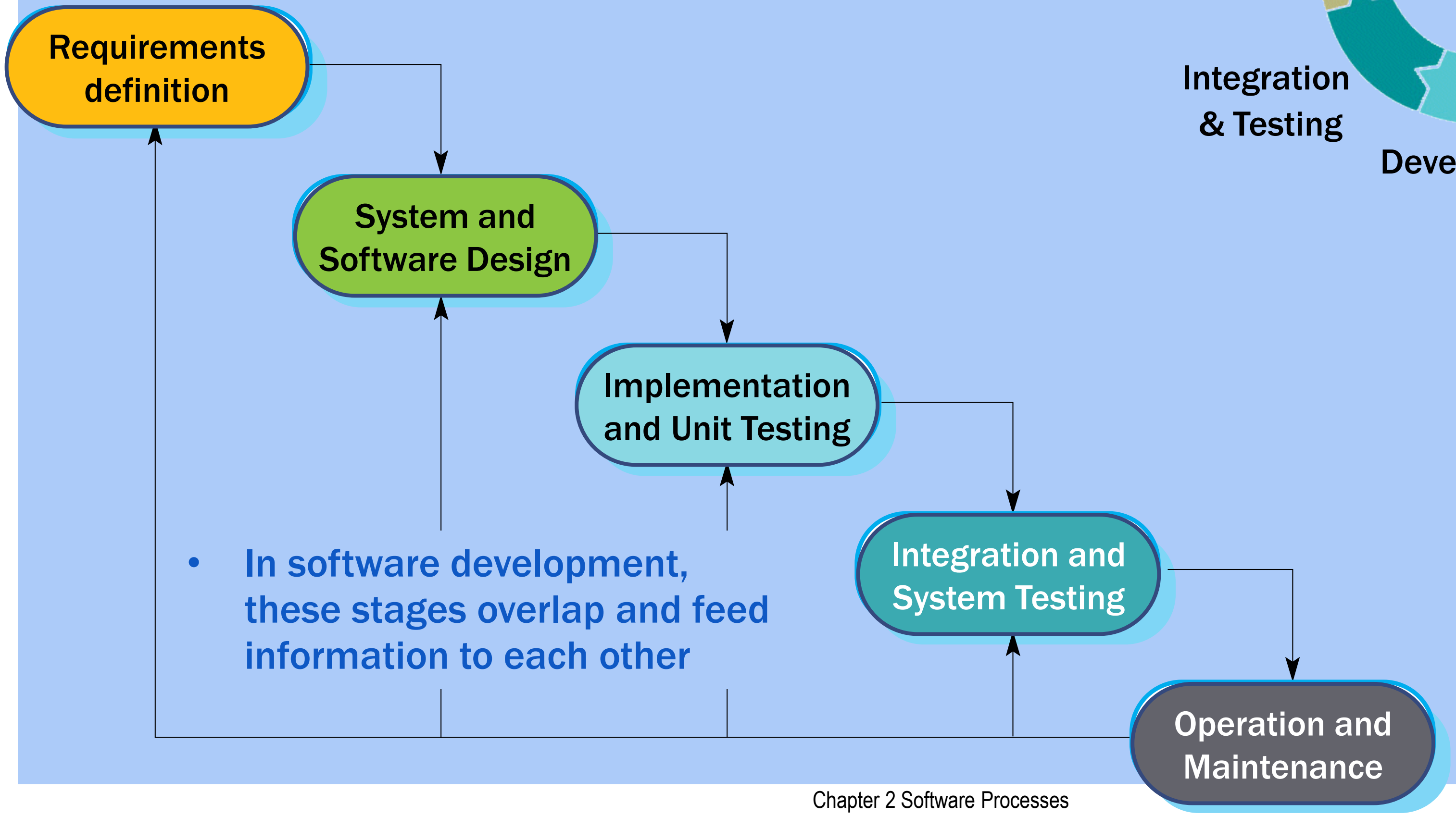
Ends upon system decommissioning

Methodologies

1. The waterfall model

- In principle, the result of each phase is one or more documents that are signed off.
- The following phase should not start until the previous phase has finished.





Requirements definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

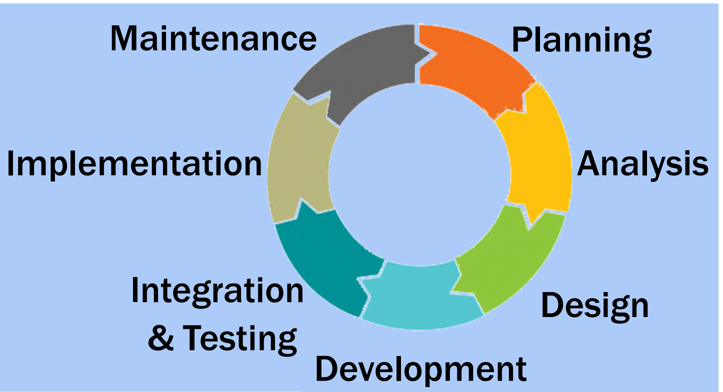
- In software development, these stages overlap and feed information to each other

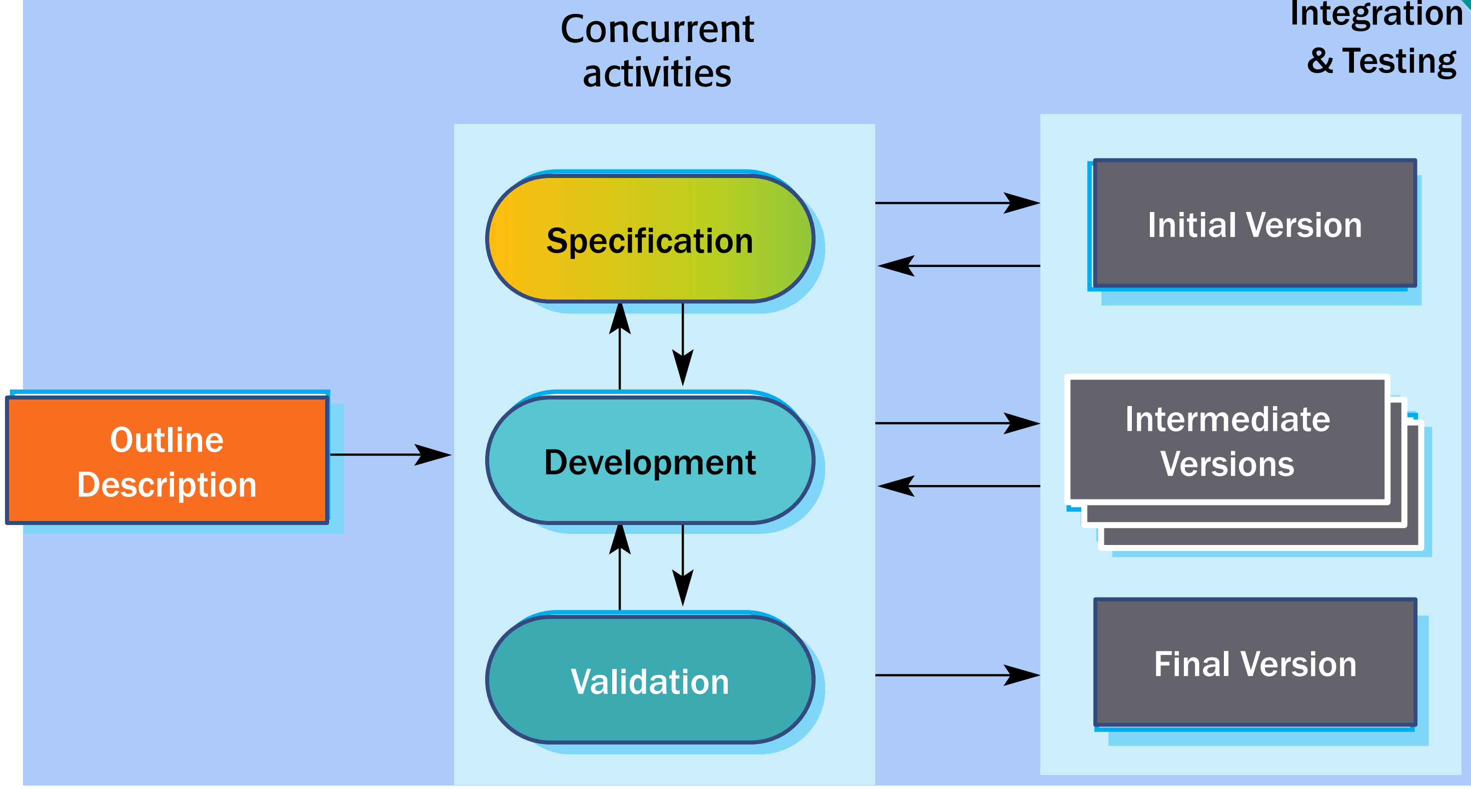
Waterfall model problems

- 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.
- Inflexible partitioning of the project into distinct stages makes it 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.
- The waterfall model is mostly used in two cases
  - Highly regulated environments where deliverables are inspected as "proof" that processes were followed rigorously.
  - 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.

2. Incremental development

Based on the idea of developing an initial implementation, getting feedback from users and others evolving the software through several versions until the required system has been developed





Agile/Scrum is a form of Incremental Development

Incremental development compared to Waterfall

Benefits

- Reduced cost to accommodate changing customer requirements
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- Easier to get customer feedback on the completed development work
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Downsides

- The process is not visible.
  - Managers often want regular deliverables to measure progress.
  - If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure may 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.
  - Incorporating further software changes becomes increasingly difficult and costly.

3. Integration and configuration based on reuse

Reusability

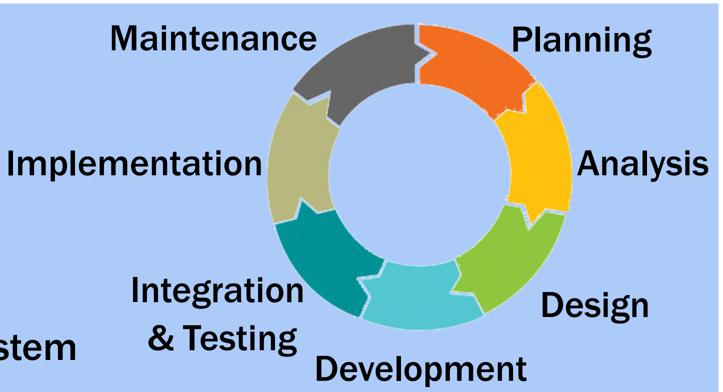
- Based on software reuse where systems are integrated from existing components or application systems (sometimes called COTS -Commercial-off-the-shelf) systems).
- Reused elements may be configured to adapt their behaviour and functionality to a user's requirements
- Reuse is now the standard approach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

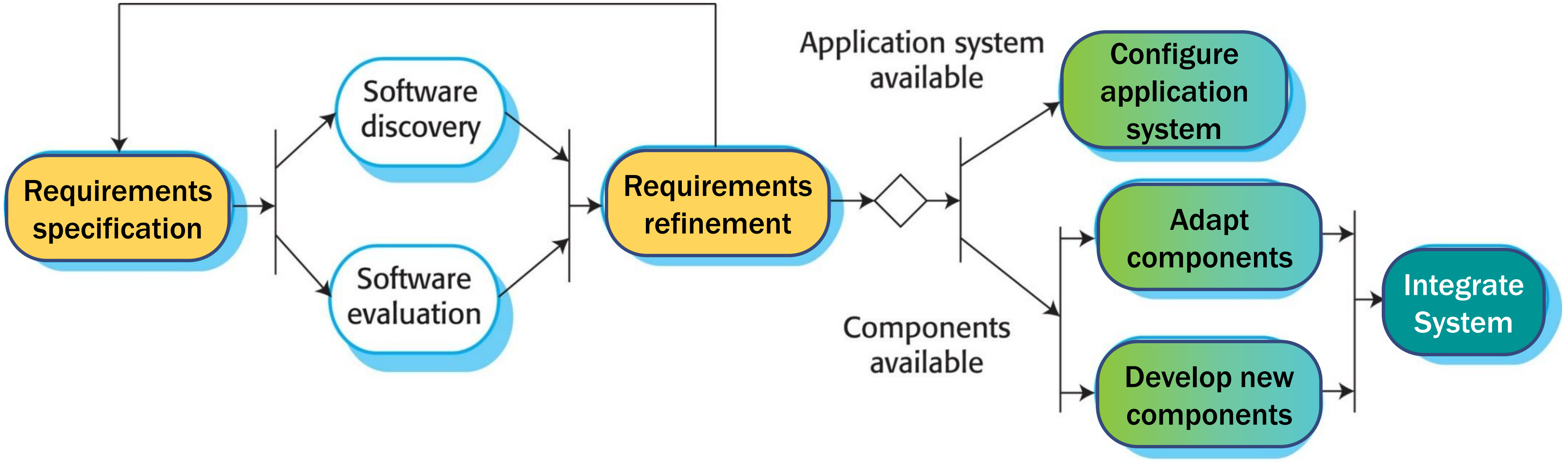
Types of Reusable Software

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

3. Integration and configuration development

- Requirements specification: initial requirements for system; description of essential requirements and desirable features
- Software discovery and evaluation: search for components, plug-ins and configuration options to meet requirements
- Requirements refinement: requirements may change based on system capability
- Application system configuration: configure the off-the-shelf system
- Component adaptation and integration: customize components using application's API.





Advantages and disadvantages of Integration and configuration development

- Reduced costs and risks as less software is developed from scratch
- Faster delivery and deployment of system
- But requirements compromises are inevitable so system may not meet real needs of users
- Loss of control over evolution of reused system elements

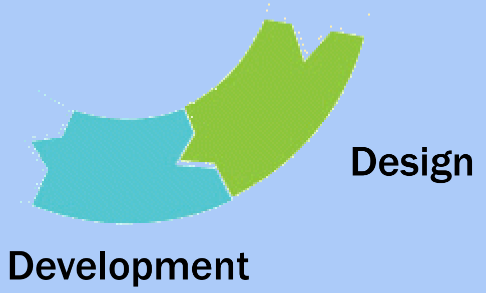
Software Processes

Software specification

- The process of establishing what services are required and the constraints on the system’s operation and development.
- Requirements engineering process
  - 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

Software design and development

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realises the specification;
- Development
  - *Note: Textbook sometimes uses the word "implementation" for the coding process – but this can be confused with the Implementation stage of the SDLC*
  - Translate this structure into an executable program;
- The activities of design and development are closely related and may be inter-leaved.
- The software is created either by developing a program or programs or by configuring an application system.
- Programming is an individual activity with no standard process.
- Debugging is the activity of finding program faults and correcting these faults.



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.
- Involves checking and review processes and system testing.
- 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.
- Testing is the most commonly used V & V activity.

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.

Coping with change

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 (e.g. re-analysing requirements) as well as the costs of implementing new functionality

Reducing the costs of rework

- Change anticipation, where the software process includes activities that can anticipate possible changes before significant rework is required.
  - For example, a prototype system may be developed to show some key features of the system to customers.
- 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.

Coping with changing requirements

- System prototyping, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This approach supports change anticipation.
- Incremental delivery, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.

Prototype development

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality
  - Prototype should focus on areas of the product that are not well-understood;
  - Error checking and recovery may not be included in the prototype;
  - Focus on functional rather than non-functional requirements such as reliability and security

Throw-away prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the 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 organisational quality standards.

Incremental development and 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
  - 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 advantages and problems

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.

Problems

- 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.
- 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.

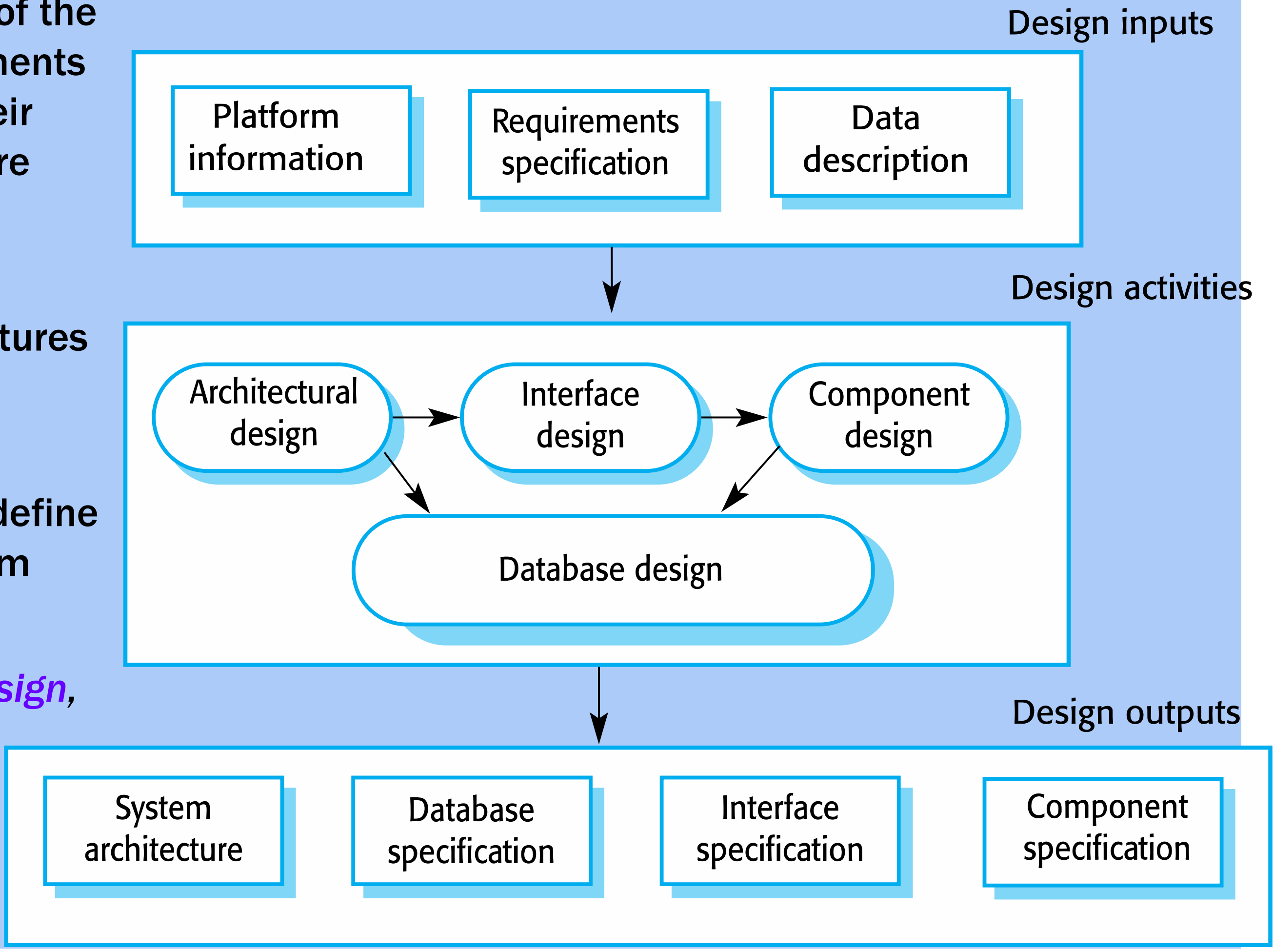
Supplemental Slides

Software process descriptions

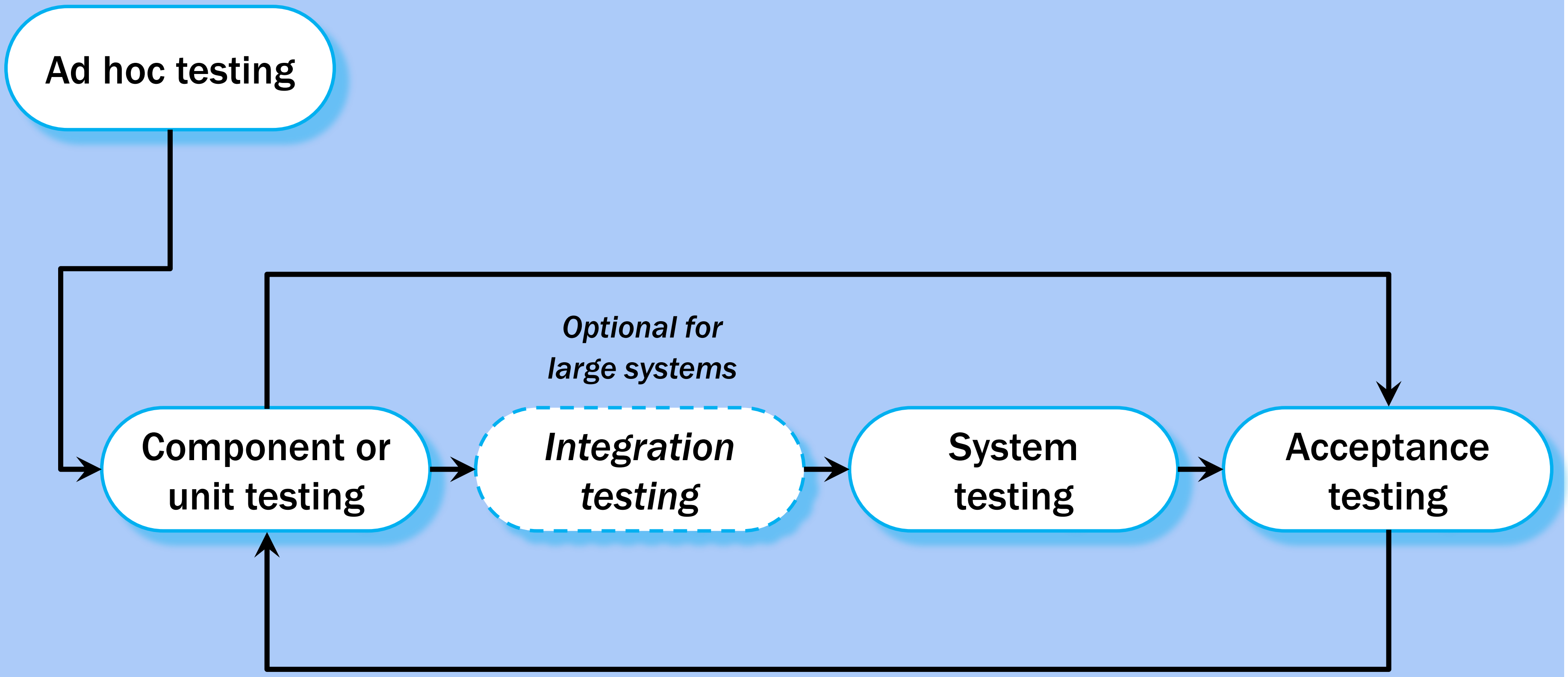
- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
- Process descriptions may also include:
  - Products, which are the outcomes of a process activity;
  - 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.

A general model of the design process and activities

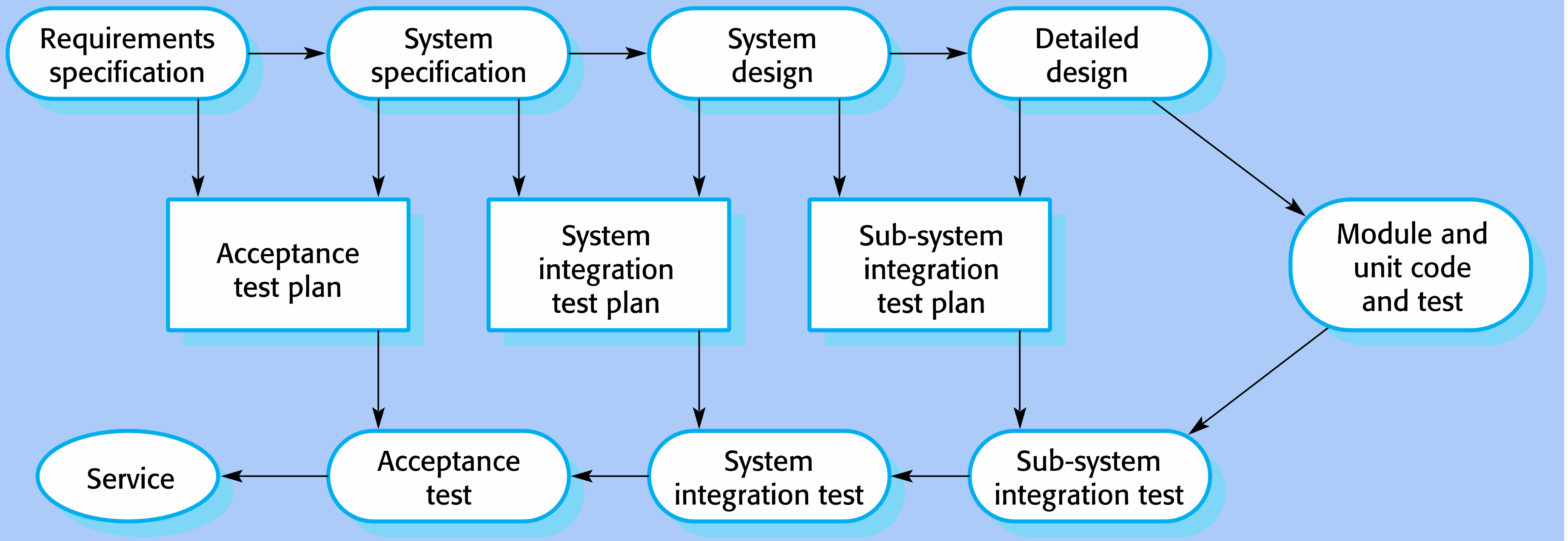
- *Architectural design*, where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.
- *Database design*, where you design the system data structures and how these are to be represented in a database.
- *Interface design*, where you define the interfaces between system components.
- *Component selection and design*, where you search for reusable components. If unavailable, you design how it will operate.



Stages of testing

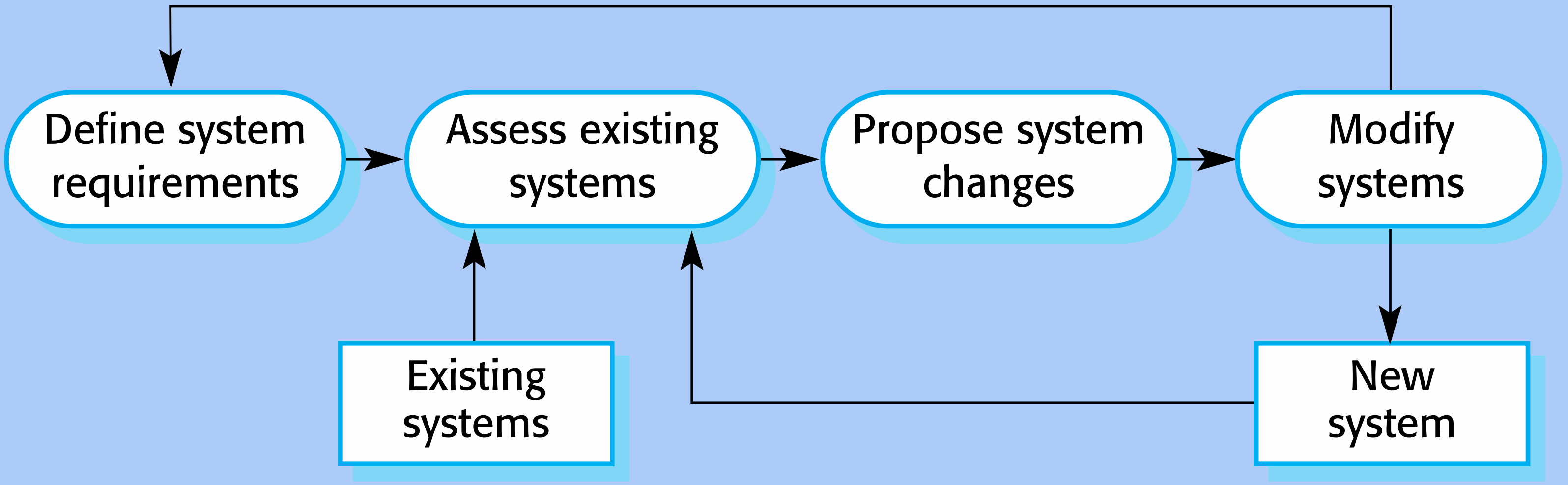


Testing phases in a plan-driven software process (V-model)



Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

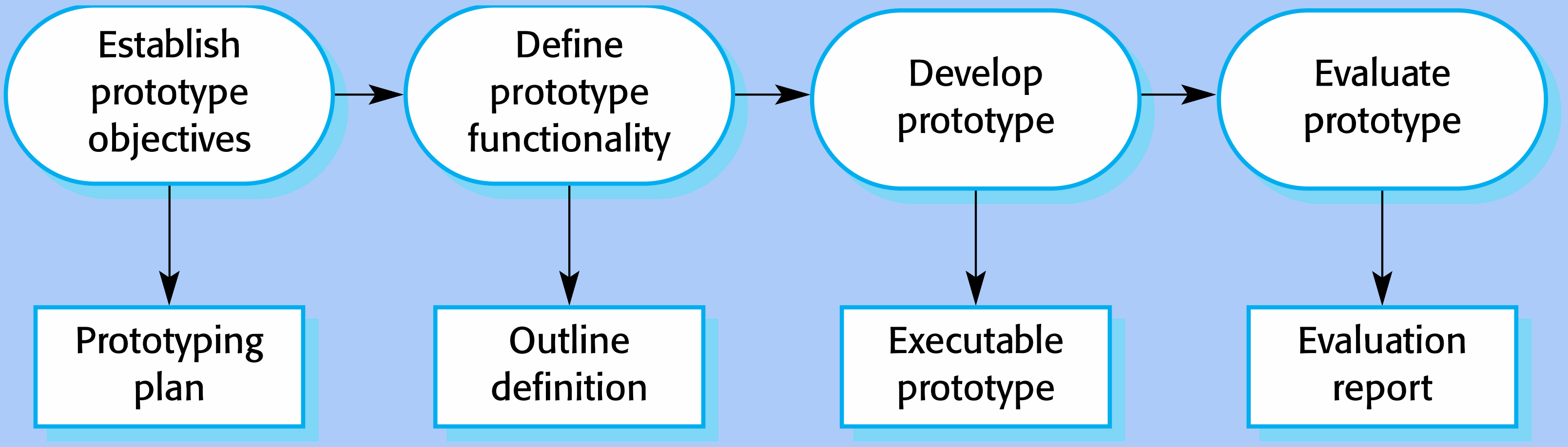
System evolution



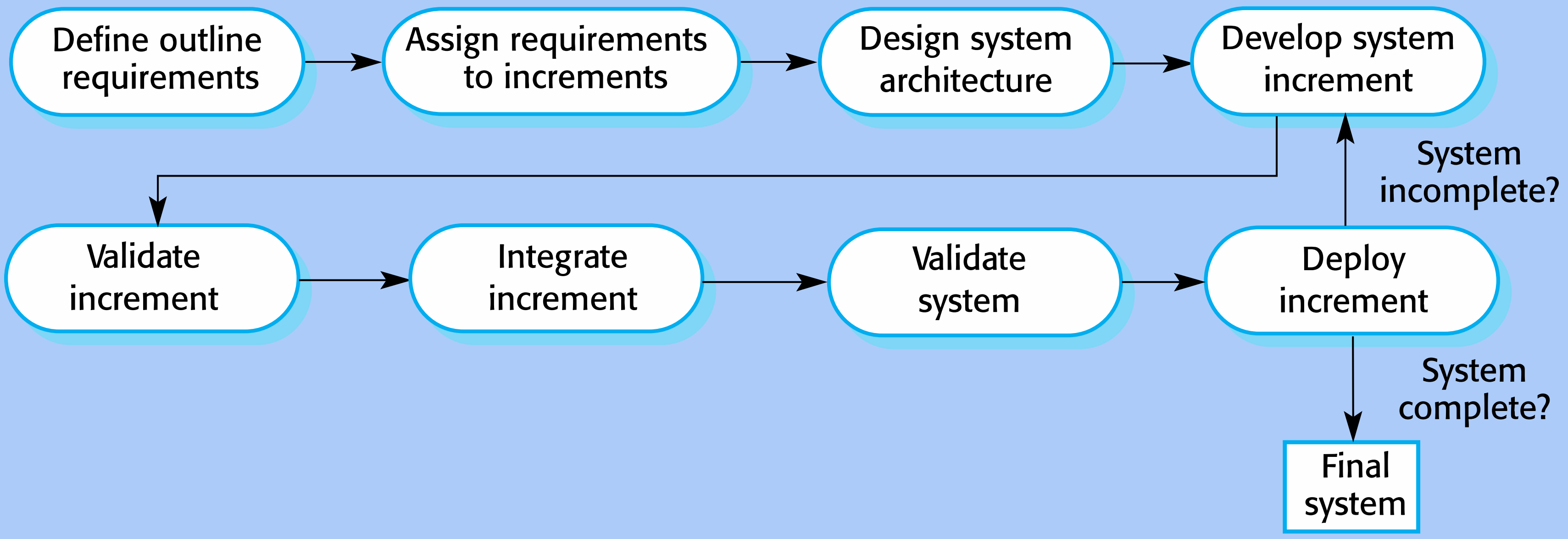
Software prototyping

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

The process of prototype development



Incremental delivery


Define outline requirements
Assign requirements to increments
Design system architecture
Develop system increment
System incomplete?
Validate increment
Integrate increment
Validate system
Deploy increment
System complete?
Final system