SWEN 6301 Software Construction

Module 2: Software Development Processes

Ahmed Tamrawi

What does the program print?

```
public class JavaPuzzle {
    public static void main(String[] args) {
        final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 * 1000;
        final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

What does the program print?

```java
public class JavaPuzzle {
    public static void main(String[] args) {
        final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 * 1000;
        final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

It prints “5”!

How to fix it?

What does the program print?

```java
public class JavaPuzzle {
    public static void main(String[] args) {
        final long MICROS_PER_DAY = 24L * 60 * 60 * 1000 * 1000;
        final long MILLIS_PER_DAY = 24L * 60 * 60 * 1000;
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);
    }
}
```

When working with large numbers, watch out for overflow—it's a silent killer.

Software Processes

A structured set of activities required to develop a software system

Topics Covered

- Software Process Models
- Process Activities
- Coping with Change
- Process Improvement

The Software Process

- 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.*
- These activities are **complex activities** in themselves, and they include sub-activities such as requirements validation, architectural design, and unit testing.
- **Processes also include other activities**, such as software configuration management and project planning that support production activities.
- 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** 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.
    - *For example, before architectural design begins, a* ***precondition*** *may be that the consumer has approved all requirements; after this activity is finished, a* ***postcondition*** *might be that the UML models describing the architecture have been reviewed.*

Software Processes

- Software processes are **complex** and, like all **intellectual** and **creative** processes rely on people making decisions and judgments.
- There is **no universal process** that is right for all kinds of software.
- Most software companies have developed their own development processes.
- Processes have evolved to take advantage of the capabilities of the software developers in an organization and the characteristics of the systems that are being developed.
- For safety-critical systems, a very **structured development process** is required where detailed records are maintained.
- For business systems, with rapidly changing requirements, a **more flexible, agile process** is likely to be better.

Plan-Driven and Agile Processes

- Professional software development is a **managed activity**, so **planning** is an inherent part of all 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 (e.g., large systems), most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

Software Process Models
sometimes called a Software Development Life Cycle or SDLC model

- **The waterfall model**

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

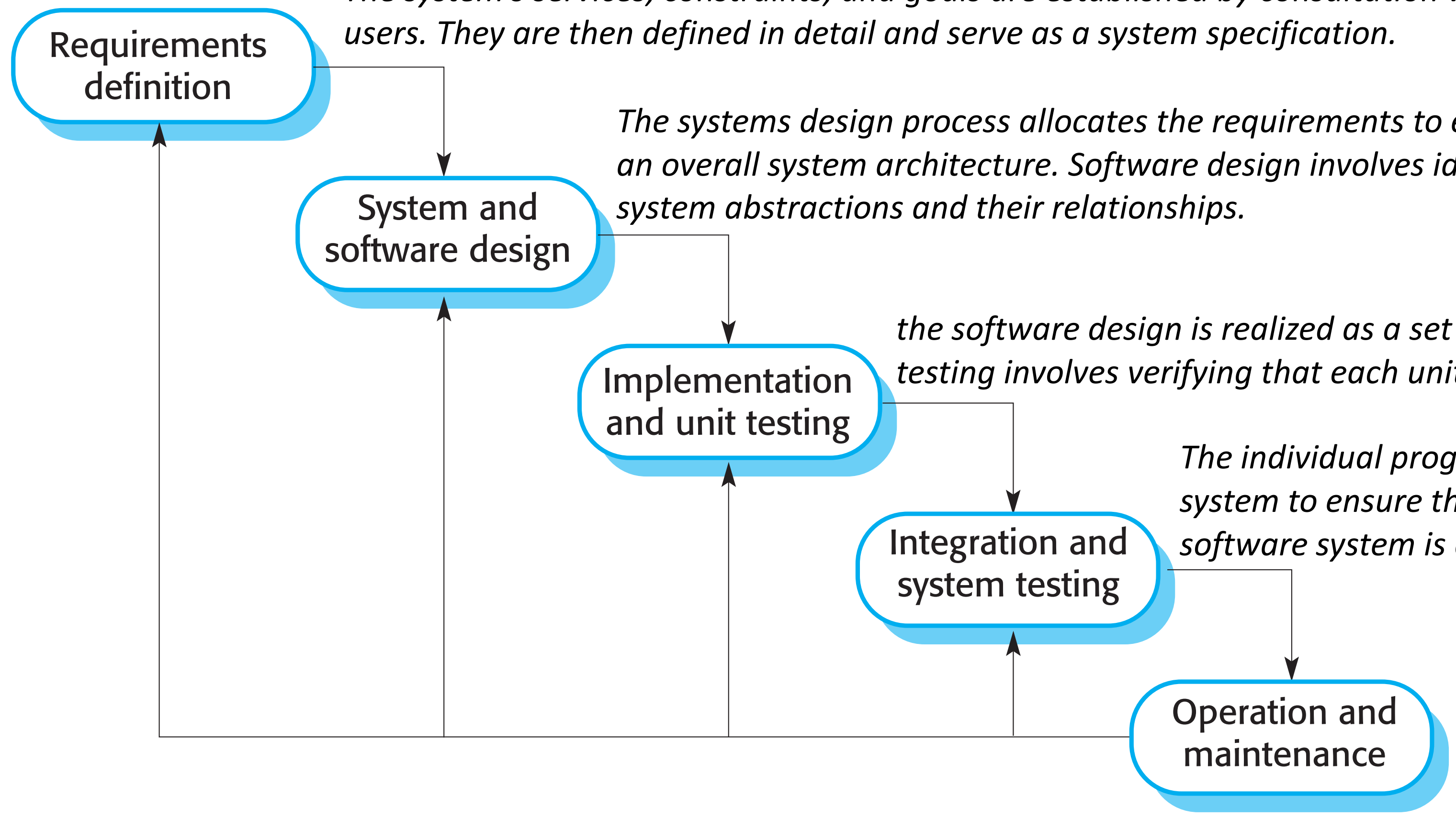
  Specification, development and validation are interleaved. May be plan-driven or agile.
- **Integration and configuration**

  The system is assembled from existing configurable (reusable) 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.

Software Process Models

- Various attempts have been made to develop "**universal**" process models that draw on all of these general models.
- One of the best known of these universal models is the **Rational Unified Process (RUP)** (Krutchen 2003), which was developed by Rational, a U.S. software engineering company.
- The RUP is a **flexible** model that can be instantiated in different ways to create processes that resemble any of the general process models discussed here.
- The RUP has been adopted by some large software companies (notably IBM), but it has not gained widespread acceptance.

The Waterfall Model


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

The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

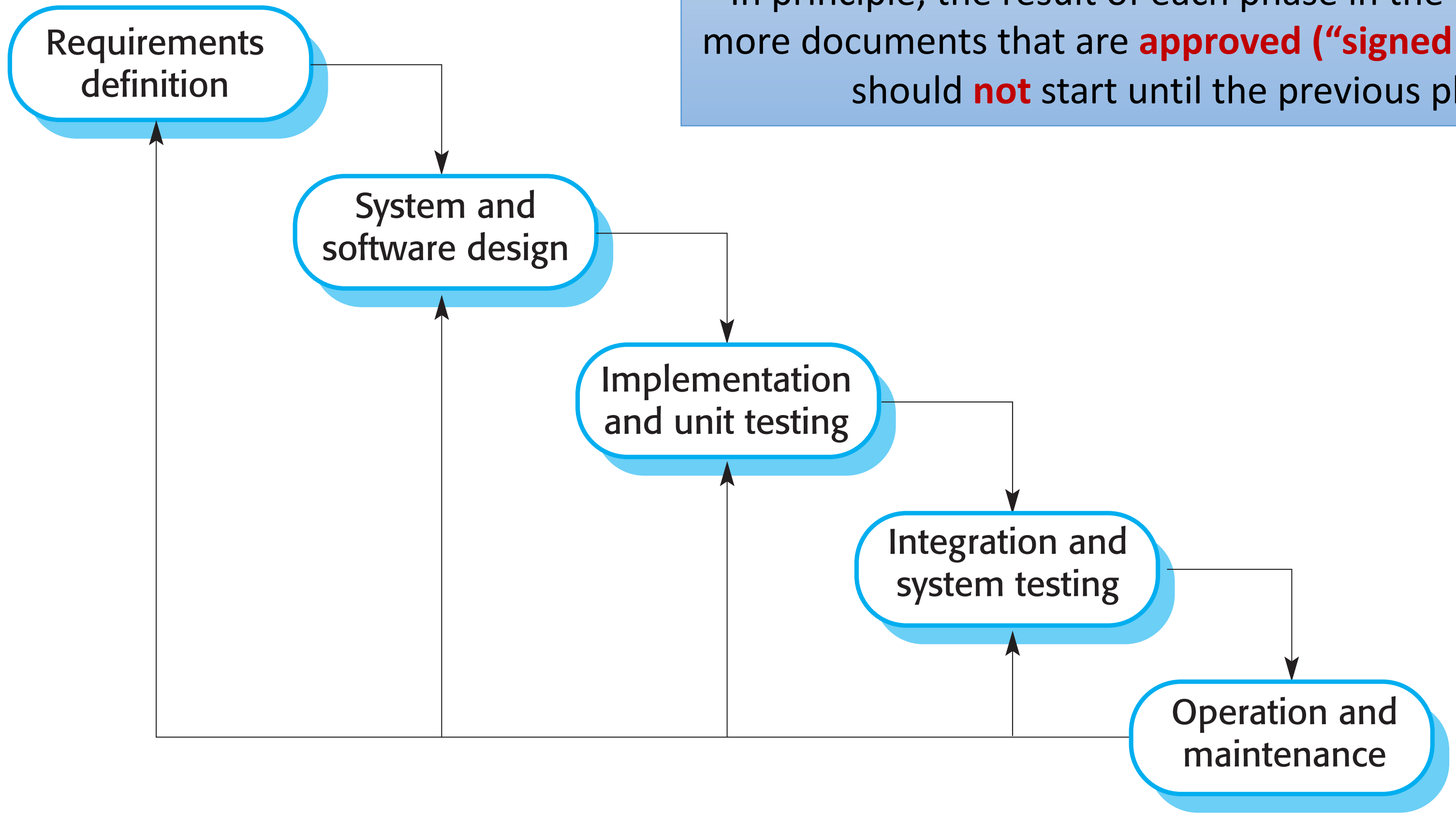
The systems design process allocates the requirements to either hardware or software systems. It establishes an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

Normally, this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors that were not discovered in earlier stages of the life cycle, improving the implementation of system units, and enhancing the system's services as new requirements are discovered.

The Waterfall Model



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

In principle, the result of each phase in the waterfall model is one or more documents that are **approved ("signed off").** The following phase should **not** start until the previous phase has finished.

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.

The Waterfall Model

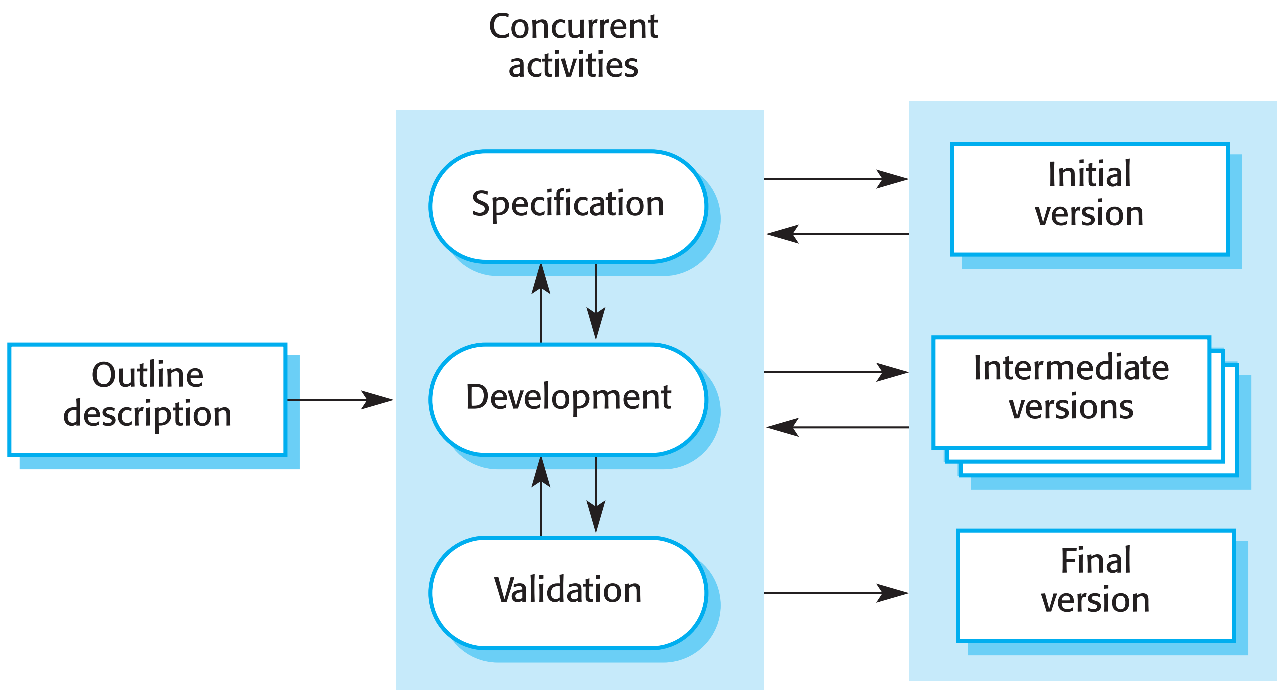
- In reality, software has to be flexible and accommodate change as it is being developed.
- The need for early commitment and system rework when changes are made means that the waterfall model is only appropriate for some types of system:
  - *Embedded systems* where the software has to interface with hardware systems (hardware inflexibility).
  - *Critical systems* where there is a need for extensive safety and security analysis of the software specification and design.
  - *Large software systems* that are part of broader engineering systems developed by several partner companies.
- *Formal Development Model* a variant of waterfall model (seL4 microkernel)

Waterfall Model Problems

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



Each increment or version of the system incorporates some of the functionality that is needed by the customer. The early increments of the system include the **most important** or most **urgently required functionality**. This means that the customer or user can evaluate the system at a relatively early stage in the development to see if it delivers what is required.

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.
- It is **easier** to get customer feedback on the development work that has been done.
  - 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 is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental Development Problems

- The process is **not visible**. (*from documentation perspective*)
  - Managers need 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 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. Incorporating further software changes becomes increasingly difficult and costly.
- The problems of incremental development become particularly **acute** for large, complex, long-lifetime systems, where different teams develop different parts of the system. Remember that Large systems need a *stable framework or architecture*.

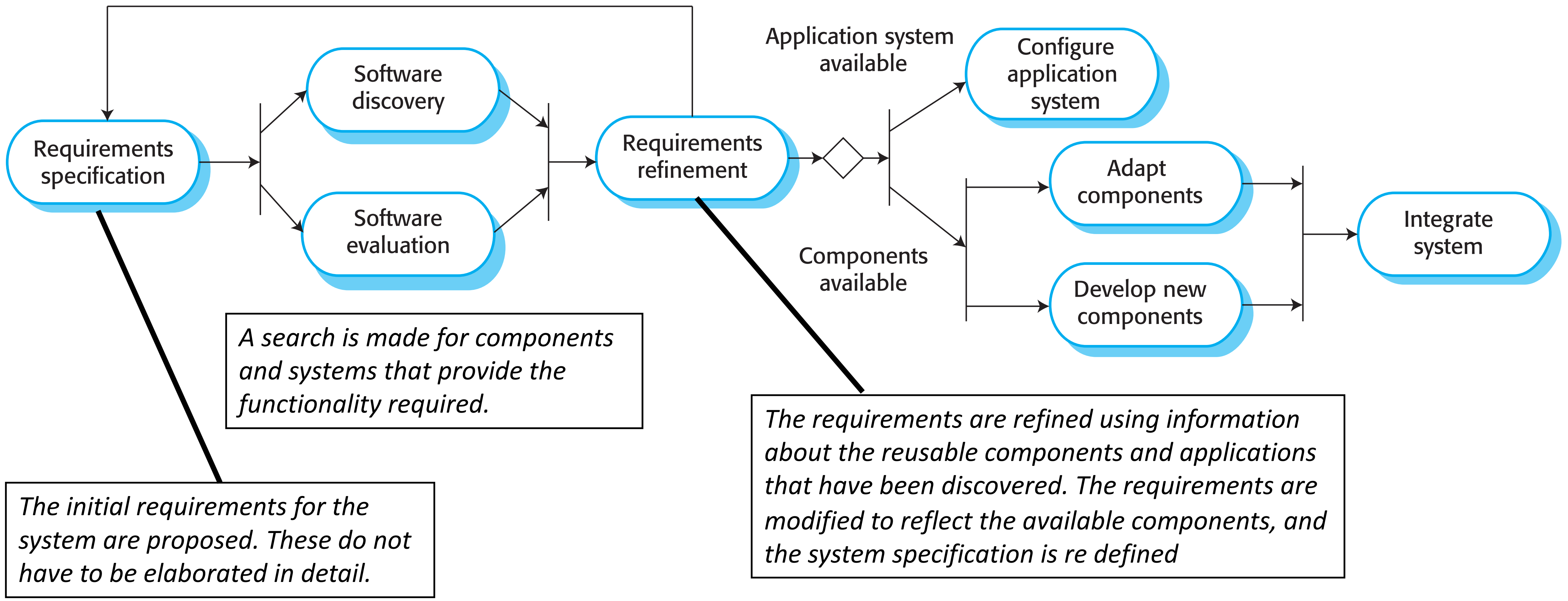
Integration and Configuration

- 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 Java Spring Framework, .NET, or J2EE.
- **Web services** that are developed according to service standards and which are available for remote invocation.

Reuse-Oriented Software Engineering



A search is made for components and systems that provide the functionality required.

The initial requirements for the system are proposed. These do not have to be elaborated in detail.

The requirements are refined using information about the reusable components and applications that have been discovered. The requirements are modified to reflect the available components, and the system specification is re defined

Reuse-Oriented Software Engineering

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

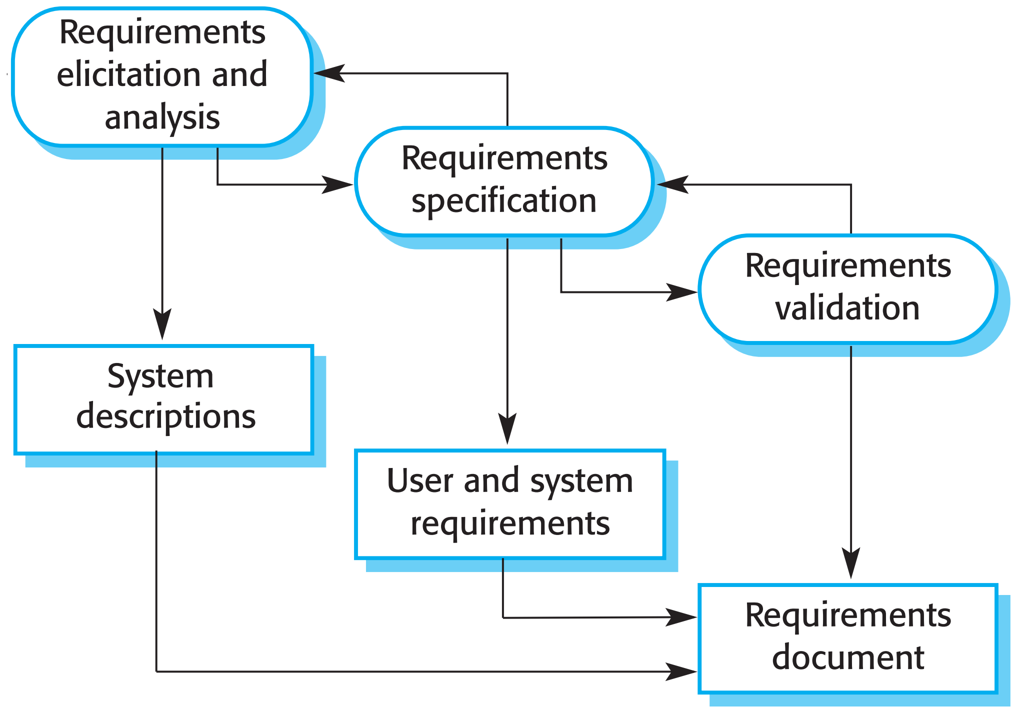
Process Activities

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.
- The four basic process activities of **specification**, **development**, **validation** and **evolution** are organized differently in different development processes.
  - For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.
- Generally, **processes are now tool-supported**.
  - This means that software developers may use a range of software tools to help them, such as requirements management systems, design model editors, program editors, automated testing tools, and debuggers.

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

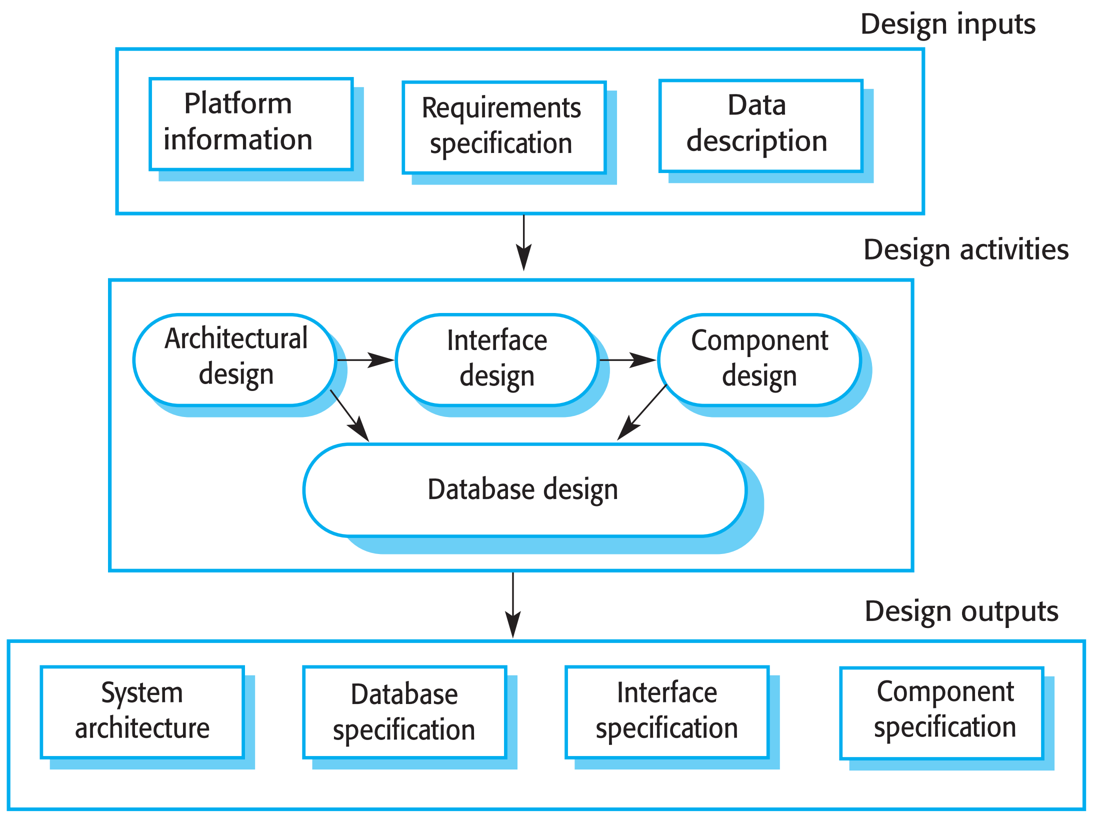

Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

Software Design and Implementation

- *The process of converting the system specification into an executable system.*
- Software design
  - Design a software structure that realises the specification;
- Implementation
  - Translate this structure into an executable program;
- The activities of design and implementation are **closely related** and may be **inter-leaved**.

Design 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.



*The design process activities are both **interleaved** and **interdependent**. New information about the design is constantly being generated, and this affects previous design decisions. **Design rework is therefore inevitable**.*

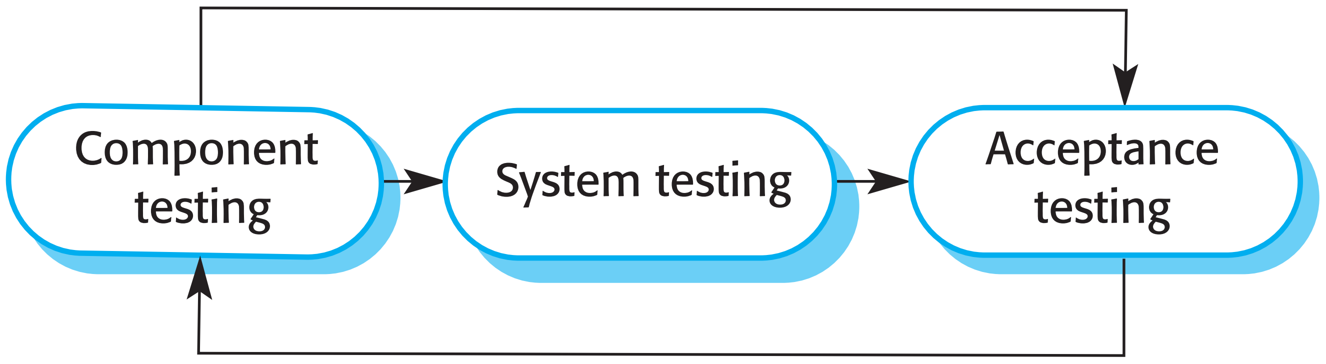
System Implementation

- The software is implemented either by *developing a program* or programs or by *configuring an application system*.
- Software development tools may be used to generate a skeleton program from a design.
- Design and implementation are **interleaved** activities for most types of software system.
- Programming is an individual activity with no standard process.
- Normally, programmers carry out some testing of the code they have developed. This often reveals program defects (bugs) that must be removed from the program. Finding and fixing program defects is called **debugging**.

Software Validation

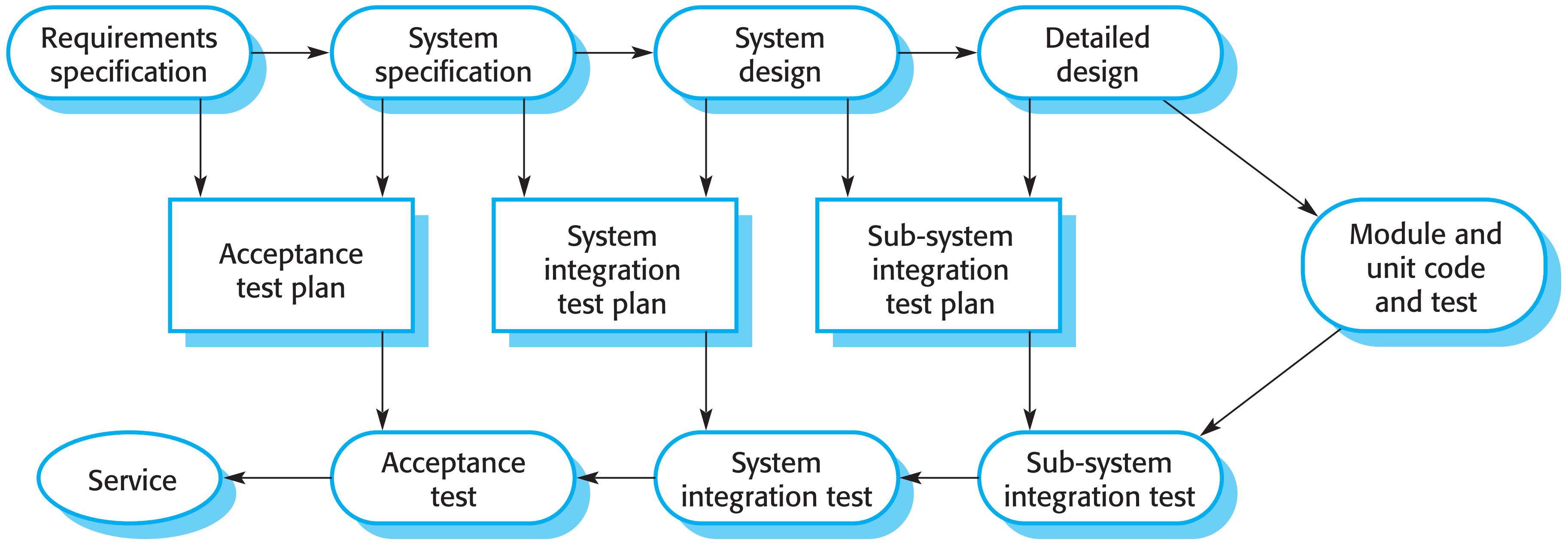
- *Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.*
- Involves **checking** and **review processes** and **system testing**.
- System testing involves *executing the system with test cases* that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.

Testing Stages



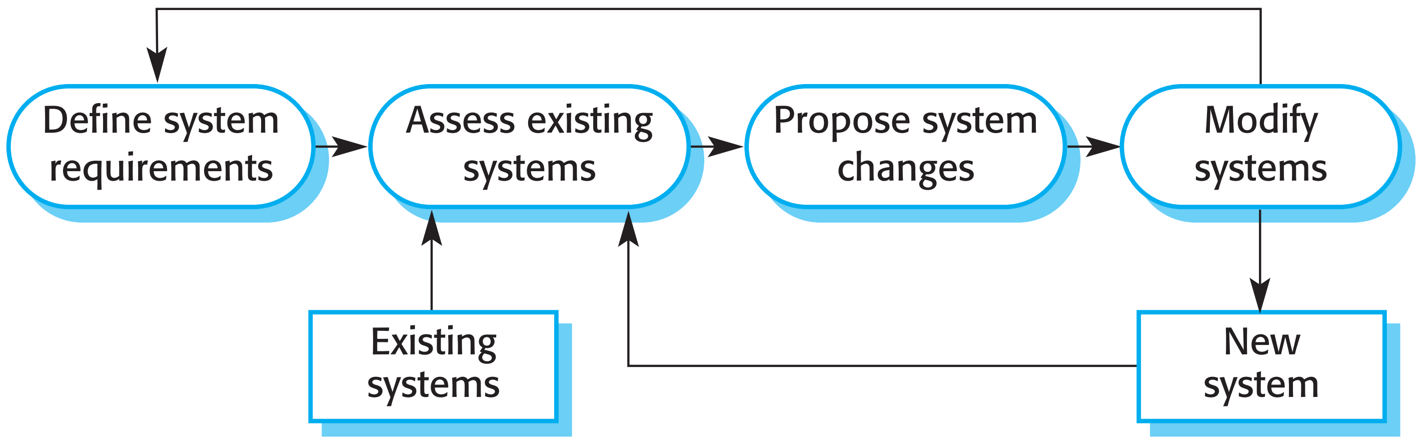
- **Component testing**
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
- **System testing**
  - Testing of the system as a whole. Testing of **emergent properties** is particularly important.
- **Customer testing**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

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

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-analyzing 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.

Software Prototyping

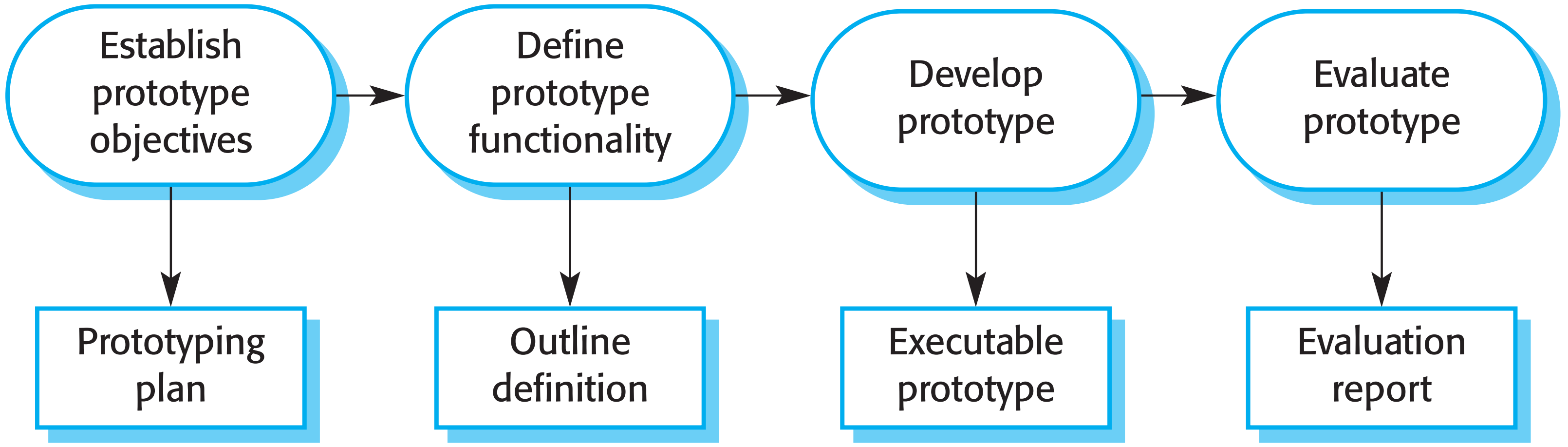
- A prototype is an ***initial*** *version of a system used to demonstrate concepts and try out design options.*
- A prototype can be used in:
  - The requirements engineering process to help with requirements elicitation and validation;
  - In design processes to explore options and develop a UI design;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

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 organizational quality standards.

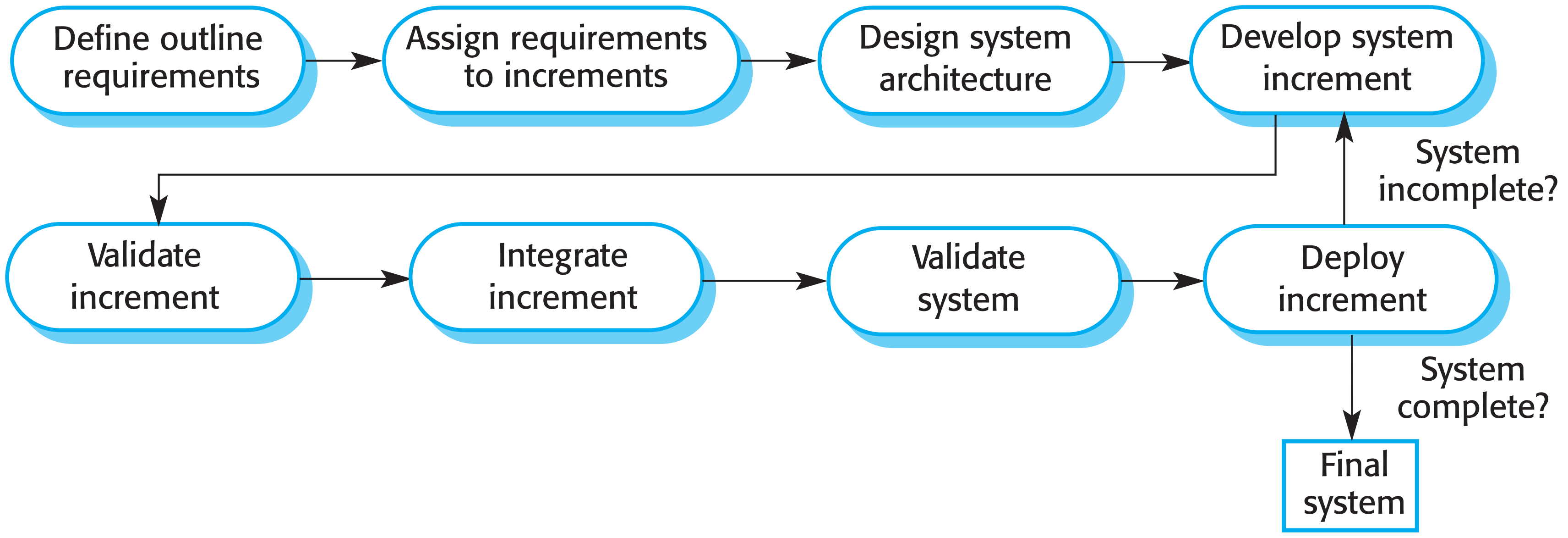
Incremental Delivery

- 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

- 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.

Process Improvement

Process Improvement

- Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes.
- **Process improvement** means *understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time*.

Approaches to Improvement

- **The process maturity approach**, which focuses on improving process and project management and introducing good software engineering practice.
  - The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes.
- **The agile approach**, which focuses on iterative development and the reduction of overheads in the software process.
  - The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.

Process Improvement Activities

- ***Process measurement***

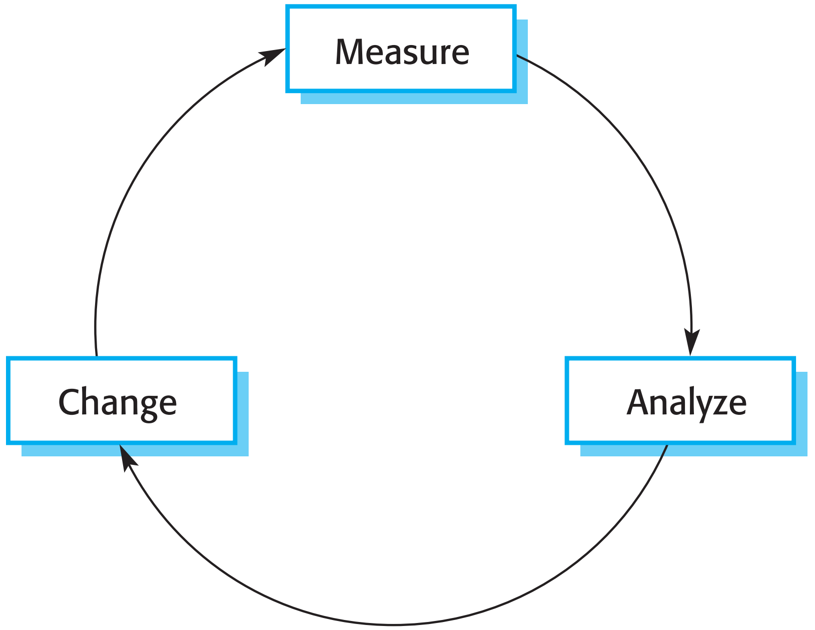
  You measure one or more attributes of the software process or product. These measurements forms a baseline that helps you decide if process improvements have been effective.

- ***Process analysis***

  The current process is assessed, and process weaknesses and bottlenecks are identified. Process models (sometimes called process maps) that describe the process may be developed.

- ***Process change***

  Process changes are proposed to address some of the identified process weaknesses. These are introduced and the cycle resumes to collect data about the effectiveness of the changes.



Process Measurement

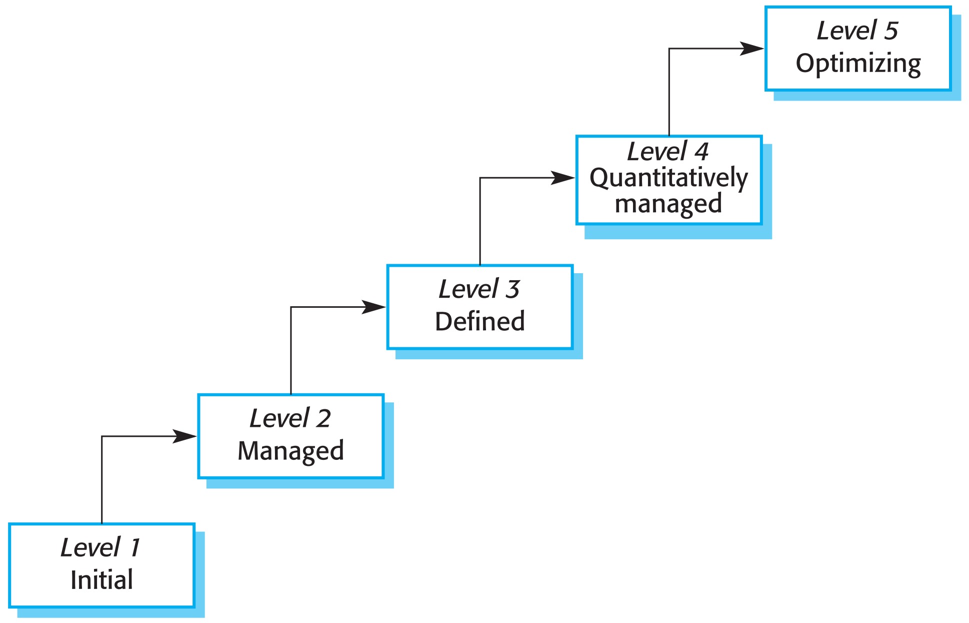
- Wherever possible, **quantitative process data** should be collected
  - However, where organizations do not have clearly defined process standards this is very difficult as you don’t know what to measure.
  - A process may have to be defined before any measurement is possible.
- Process measurements should be used to assess process improvements
  - But this does not mean that measurements should drive the improvements. The improvement driver should be the **organizational objectives**.

Process Metrics

- Time taken for process activities to be completed
  - **Example:** Calendar time or effort to complete an activity or process.
- Resources required for processes or activities
  - **Example:** Total effort in person-days.
- Number of occurrences of a particular event
  - **Example:** Number of defects discovered.

SEI Capability Maturity Levels

- Initial
  - Essentially uncontrolled
- Managed
  - Quality management strategies defined and used
- Defined
  - Process management procedures and strategies defined and used
- Repeatable
  - Product management procedures defined and used
- Optimising
  - Process improvement strategies defined and used



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.
- Requirements engineering is the process of developing a software specification.

Key Points – Cont’d

- 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.

Key Points – Cont’d

- Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.
- The principal approaches to process improvement are agile approaches, geared to reducing process overheads, and maturity-based approaches based on better process management and the use of good software engineering practice.
- The SEI process maturity framework identifies maturity levels that essentially correspond to the use of good software engineering practice.