

What Is Software Engineering?

Week 1: Introduction to Software Engineering and Software Processes

Edmund Yu, PhD

Associate Professor

esyu@syr.edu



**SYRACUSE
UNIVERSITY**
**ENGINEERING
& COMPUTER
SCIENCE**

What Is Software?

- ❖ Computer programs and associated documentation
- ❖ Software products may be:
 - ❖ Generic products: software developed for a general market
 - ❖ PC software such as graphics programs, project-management tools; CAD software; software for specific markets such as appointments systems for dentists, and so on
 - ❖ Customized products: software developed for a particular customer
 - ❖ Embedded control systems, air traffic control software, traffic monitoring systems, and so on
- ❖ Software is hard (Donald Knuth).
 - ❖ Can become extremely complex, difficult to understand, and expensive to change.
 - ❖ It's hard enough to find an error in your code when you're looking for it; it's even harder when you've assumed your code is error-free (Steve McConnell, *Code Complete*).

What Are the Attributes of Good Software?

❖ Good software:

- ❖ Should deliver the required functionality and performance to the user
- ❖ Should be maintainable, dependable, efficient and usable
 - ❖ Definitions on next slide

Attributes of Good Software

Product Characteristic	Description
Maintainability	Software should be written in such a way that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is inevitable in a changing business environment.
Dependability	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, and so on.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

What Is Software Engineering?

- ❖ Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
- ❖ Engineering discipline:
 - ❖ Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints
- ❖ All aspects of software production:
 - ❖ Not just technical process of development; also project management and the development of tools, methods, and so on to support software production
- ❖ The systematic approach used in software engineering is called a software process.

Software Costs

- ❖ Software costs often dominate computer system costs.
 - ❖ The costs of software on a PC are often greater than the hardware cost.
 - ❖ Roughly 60 percent of software costs are development costs, and 40 percent are testing costs.
- ❖ Software costs more to maintain than it does to develop.
 - ❖ For custom software, maintenance costs often exceed development costs.
 - ❖ For systems with a long life, maintenance costs may be several times development costs.
- ❖ Software engineering is concerned with **cost-effective** software development.

Fundamental SE Activities

- ❖ A software process consists of a sequence of activities that leads to the production of a software product.
- ❖ There are four fundamental activities common to all software processes.
 - ❖ **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation
 - ❖ **Software development**, where the software is designed and programmed
 - ❖ **Software validation**, where the software is checked to ensure that it is what the customer requires
 - ❖ **Software evolution**, where the software is modified to reflect changing customer and market requirements



**SYRACUSE
UNIVERSITY**
**ENGINEERING
& COMPUTER
SCIENCE**

Software Applications Types

Week 1: Introduction to Software Engineering and Software Processes

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

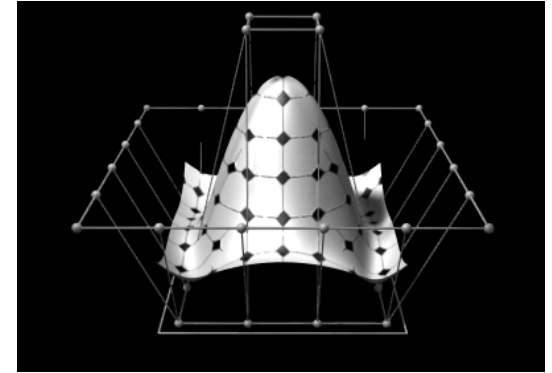


SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

Application Types

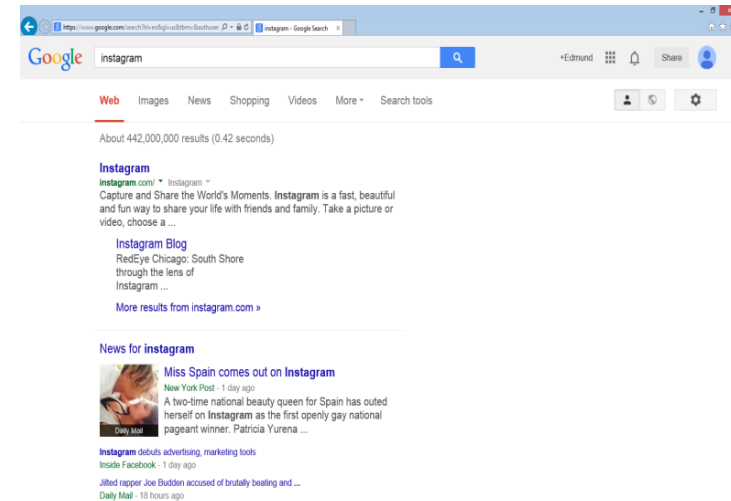
❖ Stand-alone applications

- ❖ These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.



❖ Interactive transaction-based applications

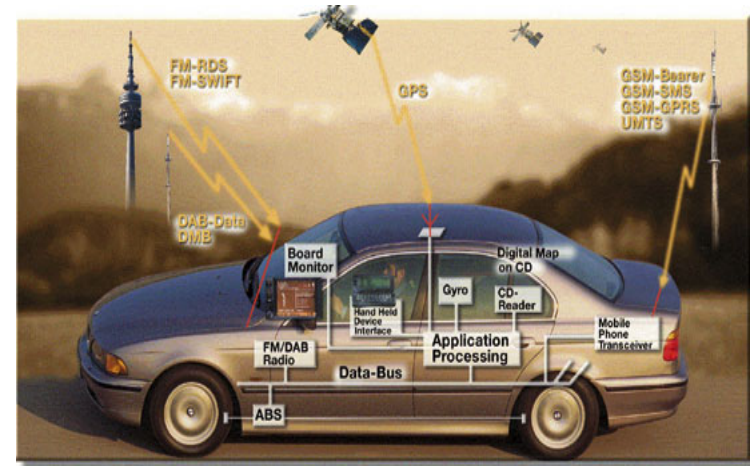
- ❖ Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.



Application Types (cont.)

❖ Embedded control systems

- ❖ These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.



❖ Batch-processing systems

- ❖ These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.

```
[java] Creating - Employee Id: 180000 Email: afaol@dde.com
[java] Creating - Employee Id: 180001 Email: xa@cxcs.com
[java] Creating - Employee Id: 180002 Email: a@gmail.com
[java] Executing Add Employees..
[java] Employee with Id: 180000 Exists: TRUE
[java] Employee with Id: 180001 Exists: TRUE
[java] Employee with Id: 180002 Exists: TRUE
[java] Creating - Employee Id: 180003 Email: afaol@dde.com
[java] Creating - Employee Id: 180004 Email: xaxs.com
[java] Creating - Employee Id: 180005 Email: a@gmail.com
[java] Executing Add Employees..
[java] Employee with Id: 180003 Exists: FALSE
[java] Employee with Id: 180004 Exists: FALSE
[java] Employee with Id: 180005 Exists: FALSE
[java] Creating - Employee Id: 180006 Email: afaol@dde.com
[java] Creating - Employee Id: 180007 Email: xa@cxcs.com
[java] Creating - Employee Id: 180008 Email:
[java] Executing Add Employees..
[java] Employee with Id: 180006 Exists: FALSE
[java] Employee with Id: 180007 Exists: FALSE
[java] Employee with Id: 180008 Exists: FALSE
[java] Creating - Employee Id: 180009 Email: la@ge.com
[java] Creating - Employee Id: 180010 Email: g@ge.org
[java] Creating - Employee Id: 180011 Email: x@x.net
[java] Executing Add Employees..
[java] Employee with Id: 180009 Exists: TRUE
[java] Employee with Id: 180010 Exists: TRUE
[java] Employee with Id: 180011 Exists: TRUE
[java] Cleanup.
```

Application Types (cont.)

❖ Entertainment systems

- ❖ These are systems that are primarily for personal use and which are intended to entertain the user.



❖ Systems for modeling and simulation

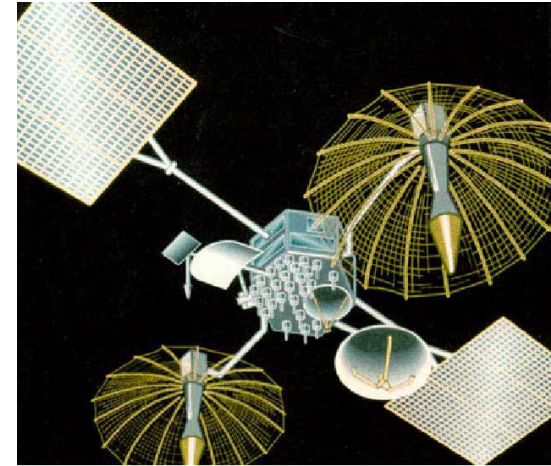
- ❖ These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.



Application Types (cont.)

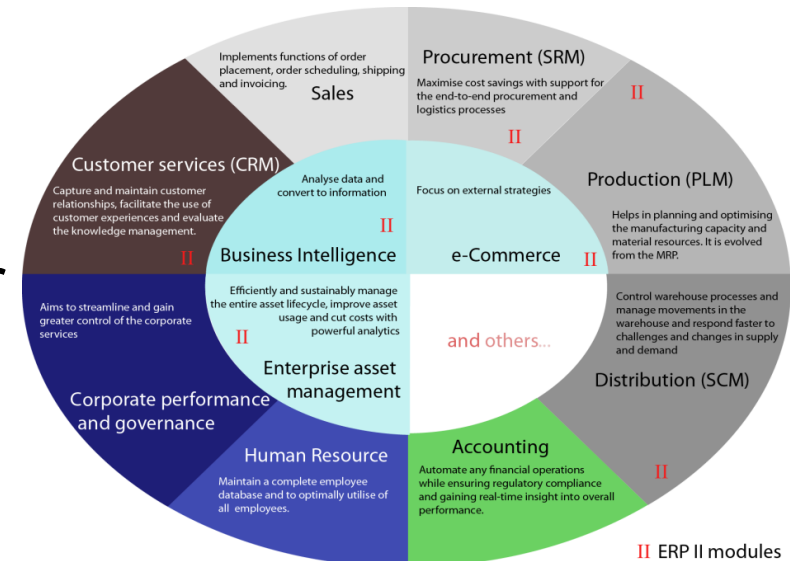
❖ Data-collection systems

- ❖ These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.



❖ Systems of systems

- ❖ These are systems that are composed of a number of other software systems.



Web Applications

- ❖ The web is now a platform for running applications, and organizations are increasingly developing web-based systems rather than local systems.
- ❖ Web services allow application functionality to be accessed over the web.
- ❖ **Cloud computing** is an approach to the provision of computer services where applications run remotely on the “cloud.”
- ❖ Three types of cloud computing:
 - ❖ Infrastructure as a service (IaaS): e.g., Amazon Elastic Compute Cloud
 - ❖ Platform as a service (PaaS): e.g., Google App Engine, Rest APIs
 - ❖ Software as a service (SaaS): e.g., Google Apps, Salesforce.com
 - ❖ Users do not buy software/hardware. They pay according to use.

Web Software Engineering

❖ **Software reuse**

- ❖ Software reuse is the dominant approach for constructing web-based systems.
 - ❖ When building these systems, you think about how you can assemble them from pre-existing software components and systems.

❖ **Incremental and agile development**

- ❖ Web-based systems should be developed and delivered incrementally.
 - ❖ It is now generally recognized that it is impractical to specify all the requirements for such systems in advance.

Web Software Engineering (cont.)

❖ **Service-oriented systems**

- ❖ Software may be implemented using service-oriented software engineering, where the software components are stand-alone web services.

❖ **Rich interfaces**

- ❖ Interface-development technologies such as AJAX and HTML5 have emerged that support the creation of rich interfaces within a web browser.



SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

A History of Software Engineering

Week 1: Introduction to Software Engineering and Software Processes

Edmund Yu, PhD
Associate Professor
esyu@syr.edu



SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

History of Software Engineering

- ❖ The term **software engineering** was suggested at conferences organized by NATO in 1968 and 1969 to discuss the **software crisis**.

“Software engineering is the establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines.”
—Peter Naur and Brian Randell (1968) at the first NATO conference of software engineering
- ❖ The software crisis was the name given to the difficulties encountered in developing large, complex systems in the 1960s.

“If builders built houses the way programmers built programs, the first woodpecker to come along would destroy civilization.”
— Gerald Weinberg in *Mastering Software Quality Assurance*

History of Software Engineering (cont.)

- ❖ It was proposed that the adoption of an engineering approach to software development would
 - ❖ Reduce the costs of software development
 - ❖ Lead to more reliable software
- “Adding manpower to a late software project makes it later.”
—Fred Brooks, *The Mythical Man-Month*
- ❖ Early 1970s:
 - ❖ Development of the notions of structured programming
 - ❖ Publication of Parnas's paper on **information hiding**
 - ❖ Development of Pascal programming language
 - ❖ Development of Smalltalk languages, which introduced notions of **object-oriented development**

Key Dates

❖ Late 1970s:

- ❖ Early use of software design methods such as Yourdon and Constantine's **structured design**

❖ Early 1980s:

- ❖ Development of the Ada programming language (structured programming + information hiding)
- ❖ Proposals for software engineering environments
- ❖ CASE tools introduced to support design methods
- ❖ Development of algorithmic approaches to software costing and estimation
- ❖ Publication of the first edition of our textbook as the first student textbook on software engineering (1982)

Key Dates (cont.)

❖ Late 1980s:

- ❖ Increased use of object-oriented programming through languages such as C++ and Objective-C
- ❖ Introduction of object-oriented design methods
- ❖ Extensive use of CASE tools

❖ Early 1990s:

- ❖ Object-oriented development becomes a mainstream development technique
- ❖ Commercial tools to support requirements engineering become available

Key Dates (cont.)

❖ Late 1990s:

- ❖ Java is developed and released in the mid-1990s.
- ❖ Increasing attention paid to notions of software architecture.
- ❖ Client-server architectures are increasingly used.
- ❖ Notion of component-based software engineering is proposed.
- ❖ The UML is proposed, integrating several separately developed notations for representing object-oriented systems.

❖ Early 2000s:

- ❖ Use of integrated development environments becomes more common.
- ❖ Use of the UML becomes widespread.
- ❖ Increasing use of scripting languages such as Python.



WIKIPEDIA
The Free Encyclopedia

Article **Talk**

Read **Edit source** View history

- Main page
- Contents
- Featured content
- Current events
- Random article
- Donate to Wikipedia
- Wikimedia Shop


- Interaction
 - Help
 - About Wikipedia
 - Community portal
 - Recent changes
 - Contact page

- Toolbox
- Print/export

- Languages
 - বাংলা
 - English

History of software engineering

From Wikipedia, the free encyclopedia



This article **needs additional citations for verification**. Please help [improve this article](#) by [adding citations to reliable sources](#). Unsourced material may be [challenged](#) and [removed](#). *(September 2011)*

From its beginnings in the 1940s, writing software has evolved into a profession concerned with how best to maximize the quality of software and of how to create it. Quality can refer to how maintainable software is, to its stability, speed, usability, testability, readability, size, cost, security, and number of flaws or "bugs", as well as to less measurable qualities like elegance, conciseness, and customer satisfaction, among many other attributes. How best to create high quality software is a separate and controversial problem covering software design principles, so-called "best practices" for writing code, as well as broader management issues such as optimal team size, process, how best to deliver software on time and as quickly as possible, work-place "culture," hiring practices, and so forth. All this falls under the broad rubric of [software engineering](#).

Contents [\[hide\]](#)

1 Overview

History of computing

Hardware

- [Hardware before 1960](#) ·
- [Hardware 1960s to present](#) ·
- [Hardware in Soviet Bloc countries](#)

Computer science

- [Artificial intelligence](#) · [Compiler construction](#) ·
- [Computer science](#) · [Operating systems](#) ·
- [Programming languages](#) ·

Software engineering

Modern concepts

- [Graphical user interface](#) · [Internet](#) ·
- [Personal computers](#) · [Laptops](#) ·
- [Video games](#) · [World Wide Web](#)



SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

Software Process Models

Week 1: Introduction to Software Engineering and Software Processes

Edmund Yu, PhD
Associate Professor
esyu@syr.edu



SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

Software Processes

- ❖ The systematic approach used in software engineering is called a software process.

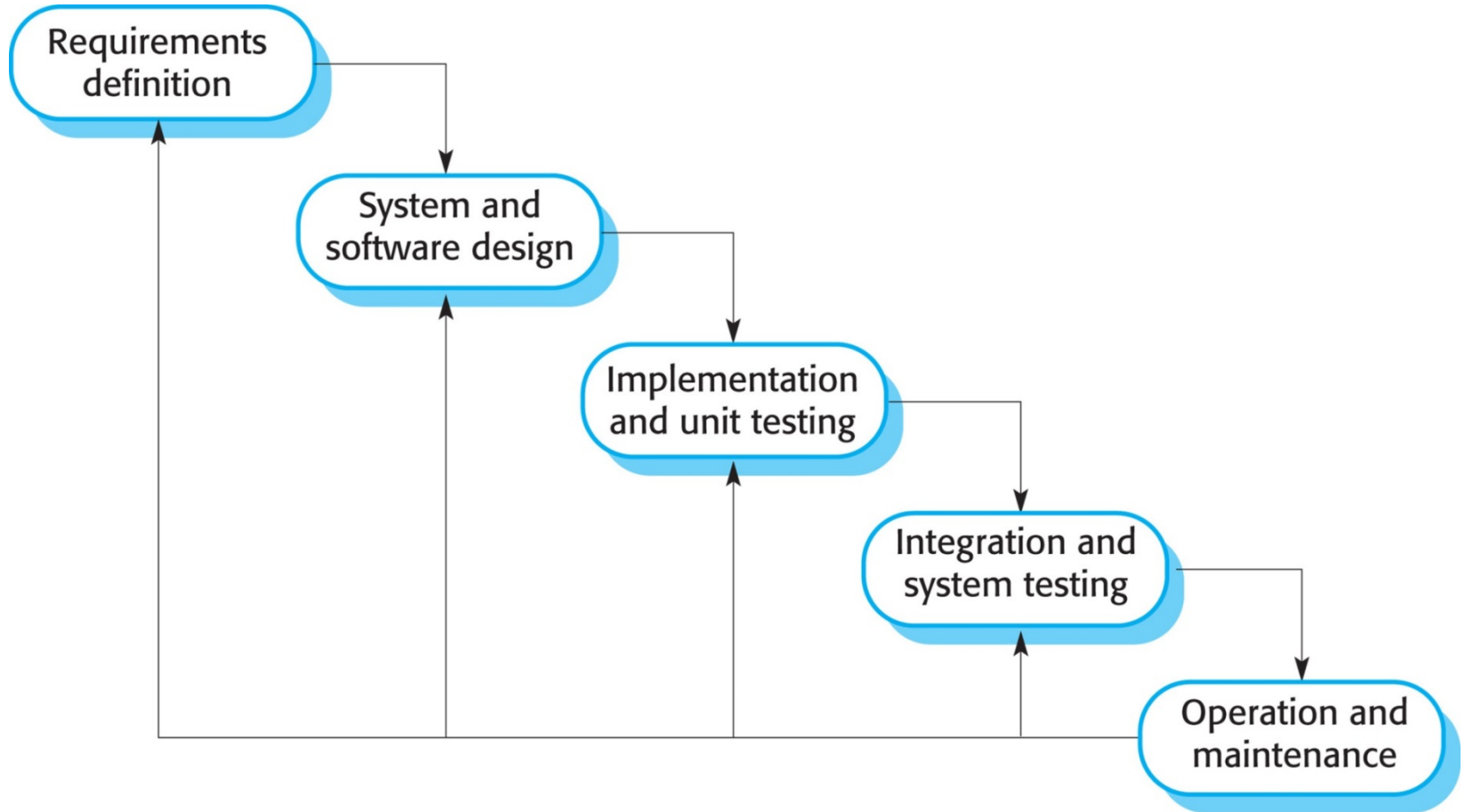
Plan-Driven and Agile Processes

- ❖ There are two main categories of software processes.
 - ❖ **Plan-driven processes**
 - ❖ Processes where all of the process activities are planned in advance, and progress is measured against this plan.
 - ❖ **Agile processes**
 - ❖ Processes where 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.

Software Process Models

- ❖ A **software process model** is a simplified representation of a software process.
- ❖ Three commonly used 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** (integration and configuration)
 - ❖ The system is assembled from existing components.
 - ❖ May be plan-driven or agile.

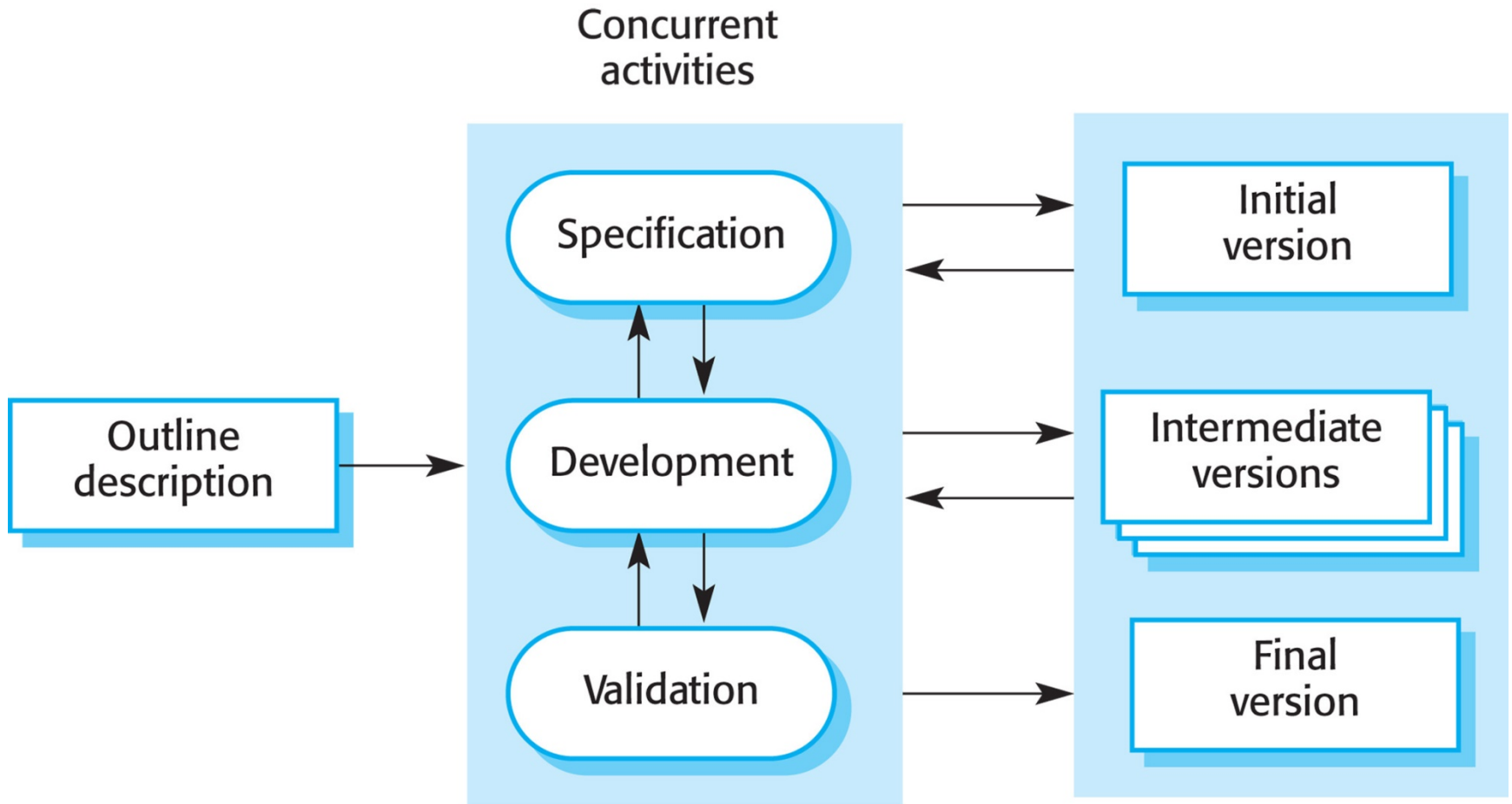
The Waterfall Model



Waterfall Model Problems

- ❖ Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
 - ❖ This model is only appropriate when
 - ❖ The requirements are well-understood.
 - ❖ Changes will be fairly limited during the design process.
- ❖ Few business systems have stable requirements.
- ❖ Therefore, 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



Software Process Models (cont.)

- ❖ A software process model is a simplified representation of a software process.
- ❖ Three commonly used 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.

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.

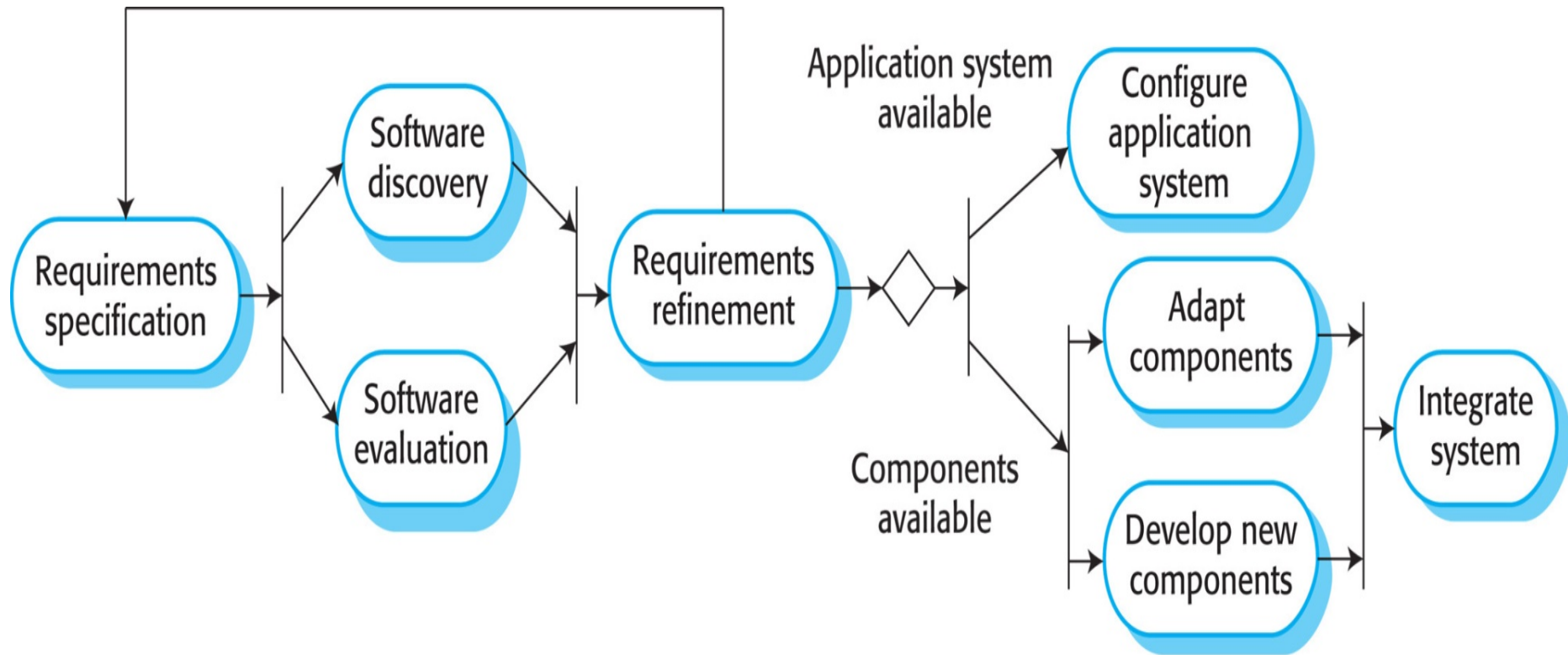
Incremental Development Problems

- ❖ The process is not visible.
 - ❖ 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 are spent on refactoring to improve the software, regular change tends to corrupt its structure.
 - ❖ Incorporating further software changes becomes increasingly difficult and costly.

Reuse-Based Software Engineering

- ❖ 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 functionality to a user's requirements.
- ❖ Reuse is now the standard approach for building many types of systems.
 - ❖ Reuse covered in more depth in chapter 15.

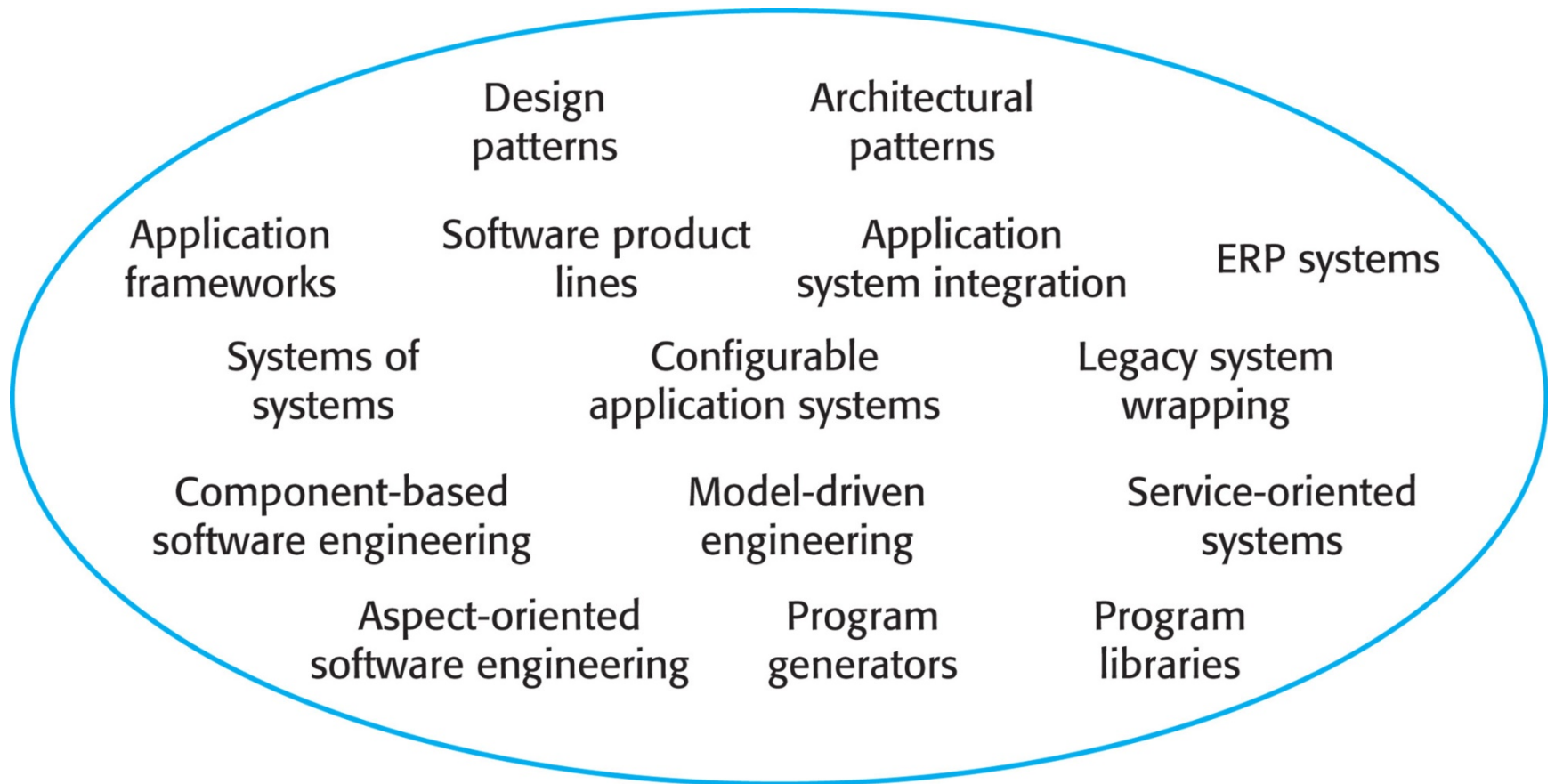
Reuse-Oriented Software Engineering



Software Process Models (cont.)

- ❖ A software process model is a simplified representation of a software process.
- ❖ Three commonly used 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.

The Reuse Landscape





SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

Software Process Activities

Week 1: Introduction to Software Engineering and Software Processes

Edmund Yu, PhD
Associate Professor
esyu@syr.edu



SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

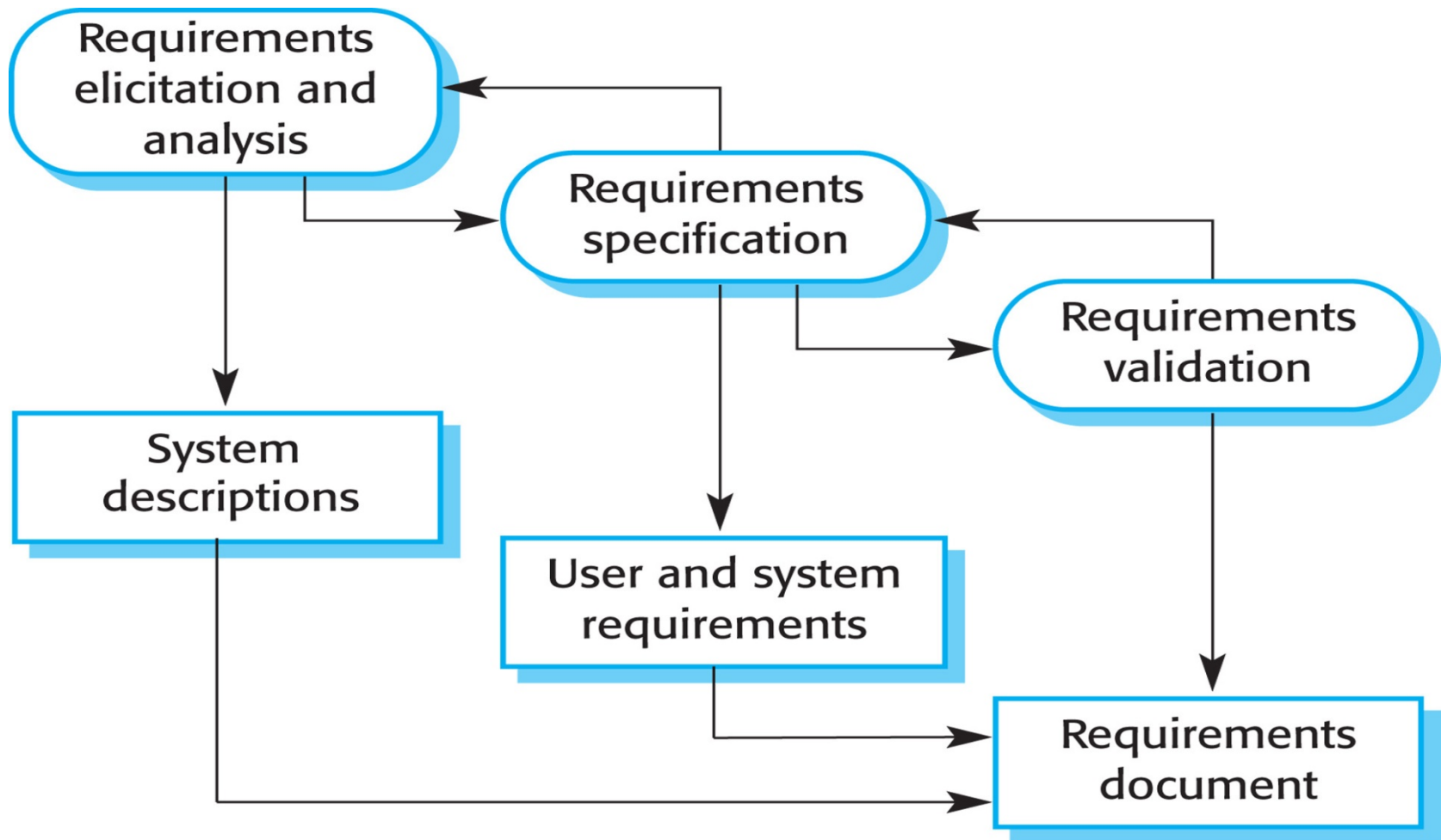
Process Activities

- ❖ A software process consists of a sequence of activities that leads to the production of a software product.
- ❖ There are four fundamental activities common to all software processes.
 - ❖ **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation
 - ❖ **Software development**, where the software is designed and programmed
 - ❖ **Software validation**, where the software is checked to ensure that it is what the customer requires.
 - ❖ **Software evolution**, where the software is modified to reflect changing customer and market requirements
- ❖ In the waterfall model, they are organized in sequence.
- ❖ In incremental development they are interleaved.

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



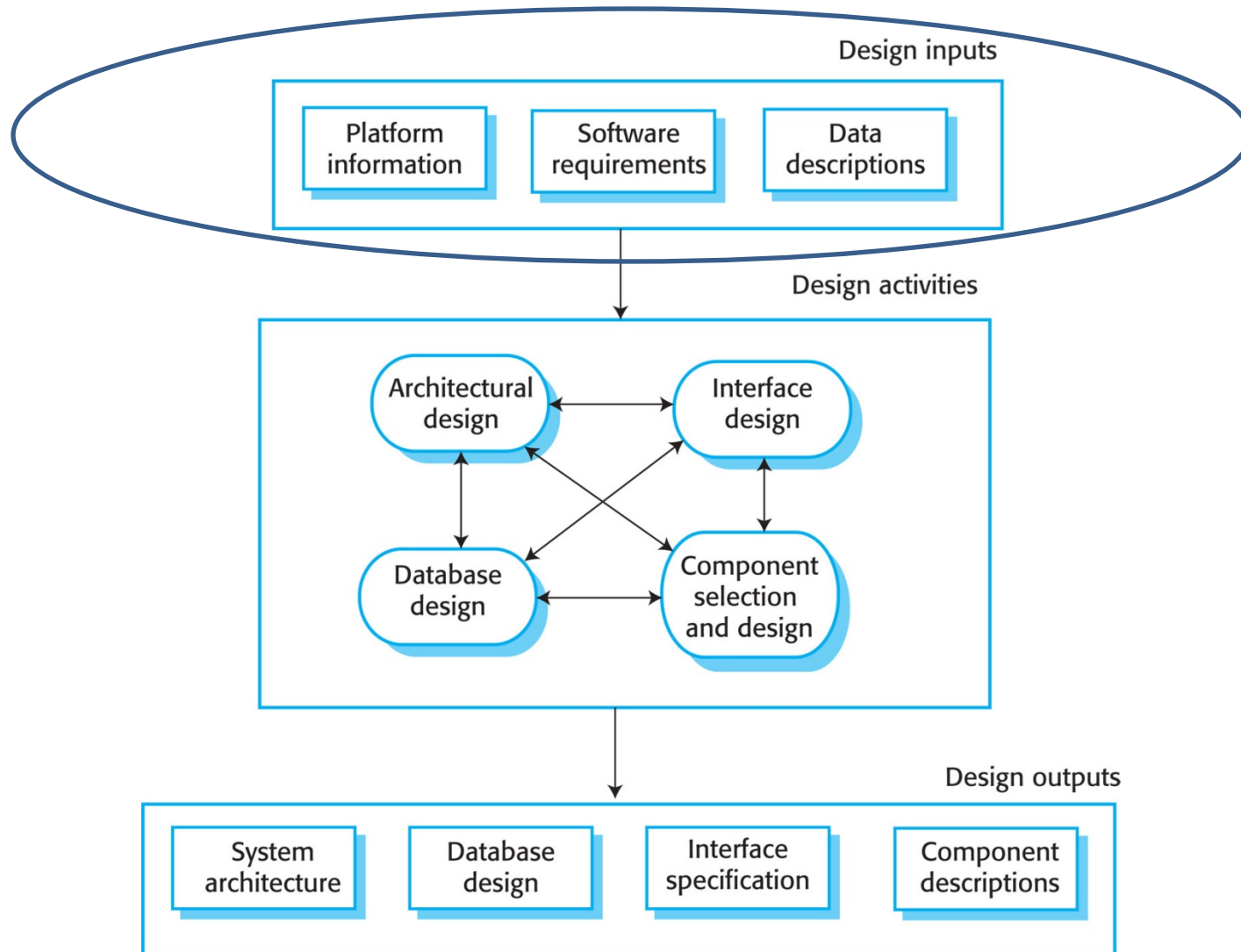
Process Activities (cont.)

- ❖ A software process consists of a sequence of activities that leads to the production of a software product.
- ❖ There are four fundamental activities common to all software processes.
 - ❖ **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation
 - ❖ **Software development**, where the software is designed and programmed
 - ❖ **Software validation**, where the software is checked to ensure that it is what the customer requires
 - ❖ **Software evolution**, where the software is modified to reflect changing customer and market requirements
- ❖ In the waterfall model, they are organized in sequence.
- ❖ In incremental development they are interleaved.

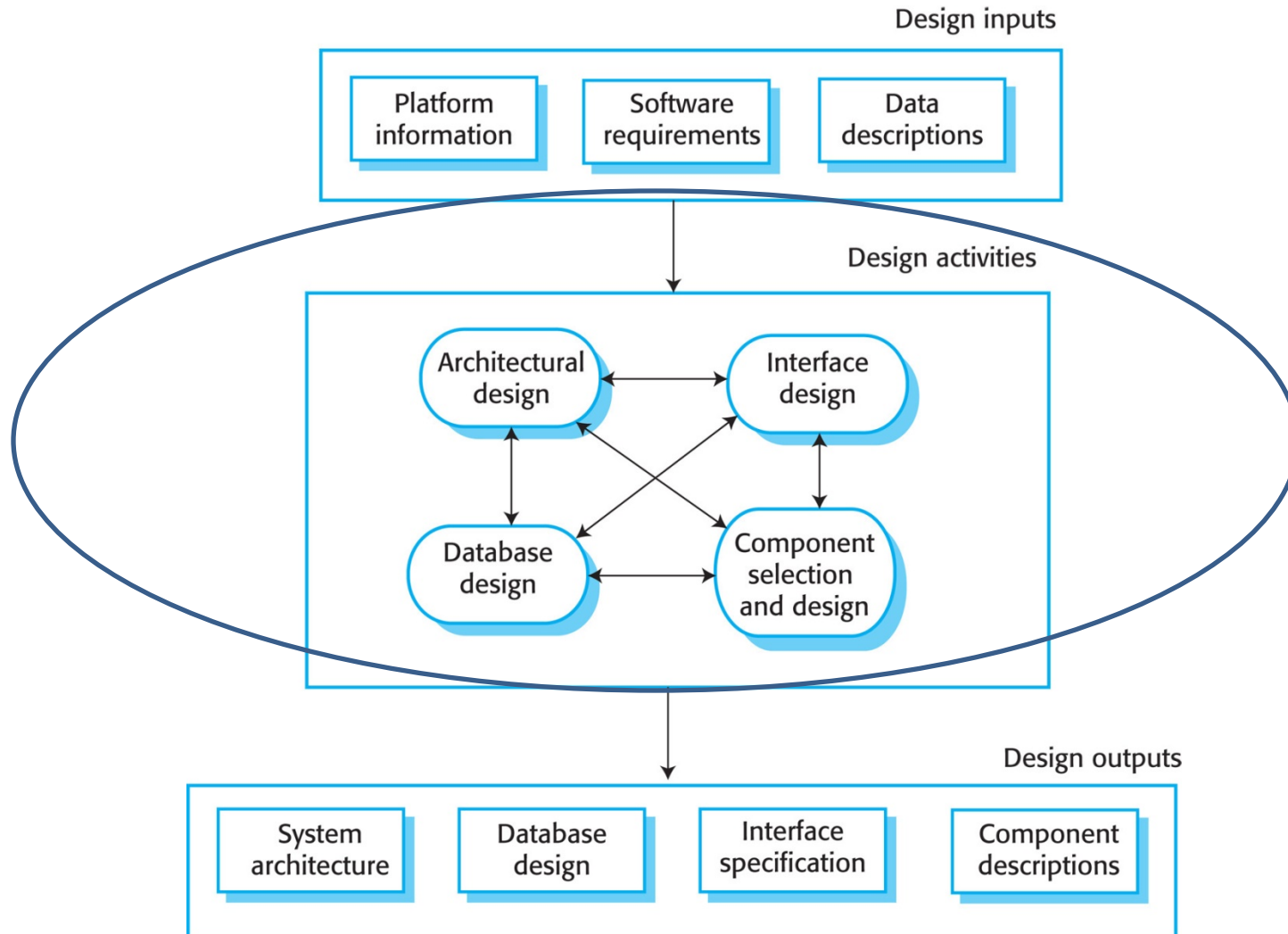
Software Development

- ❖ The process of converting the system requirements specification into an executable system.
- ❖ It includes:
 - ❖ **Software design**
 - ❖ Design a software structure that realizes the specification.
 - ❖ **Software implementation**
 - ❖ Translate this structure into an executable program.

A General Model of the Design Process

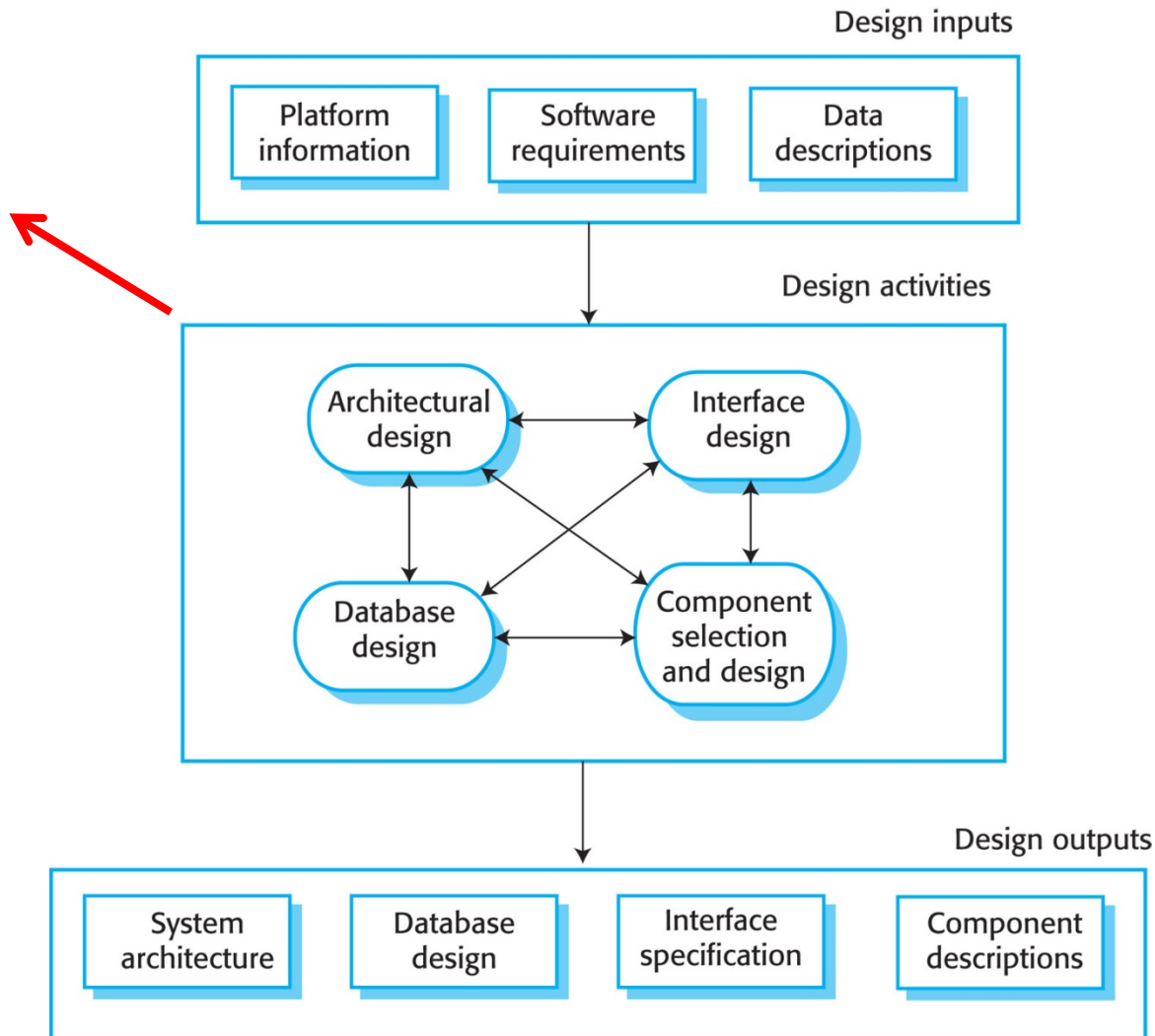


A General Model of the Design Process

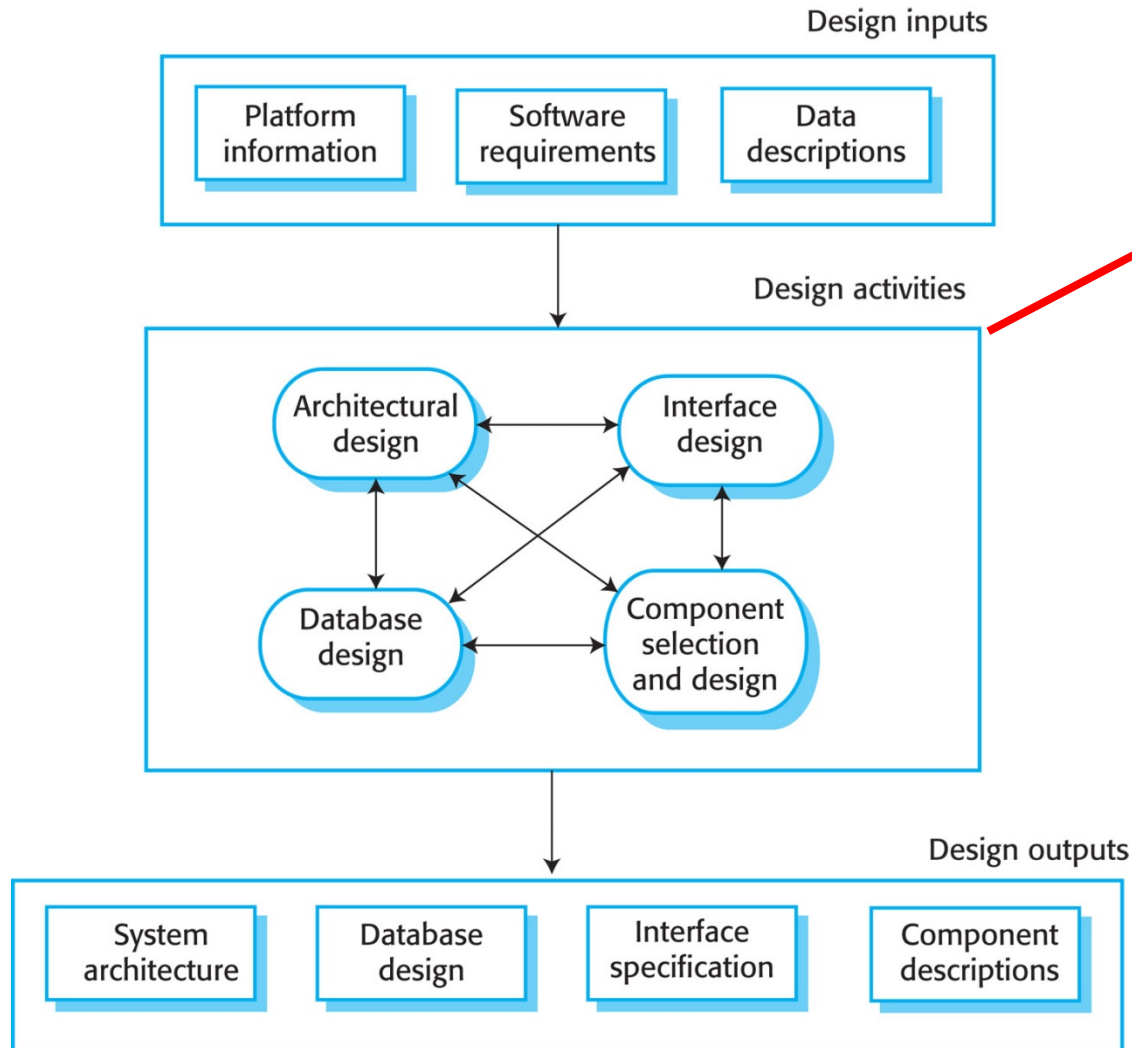


A General Model of the Design Process

Identify the overall structure of the system, its principal components (sometimes called subsystems or modules), their relationships, and how they are distributed.

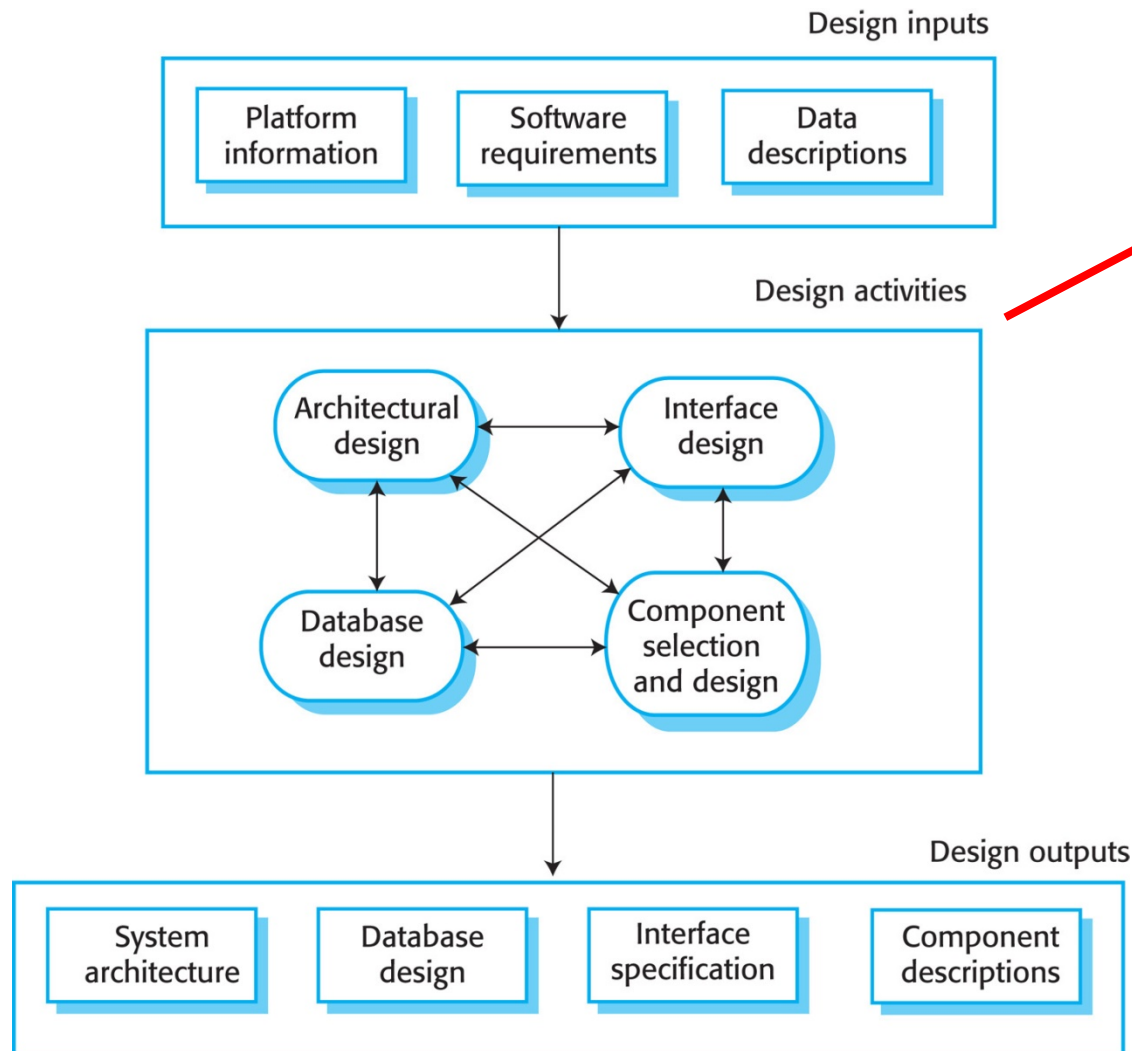


A General Model of the Design Process



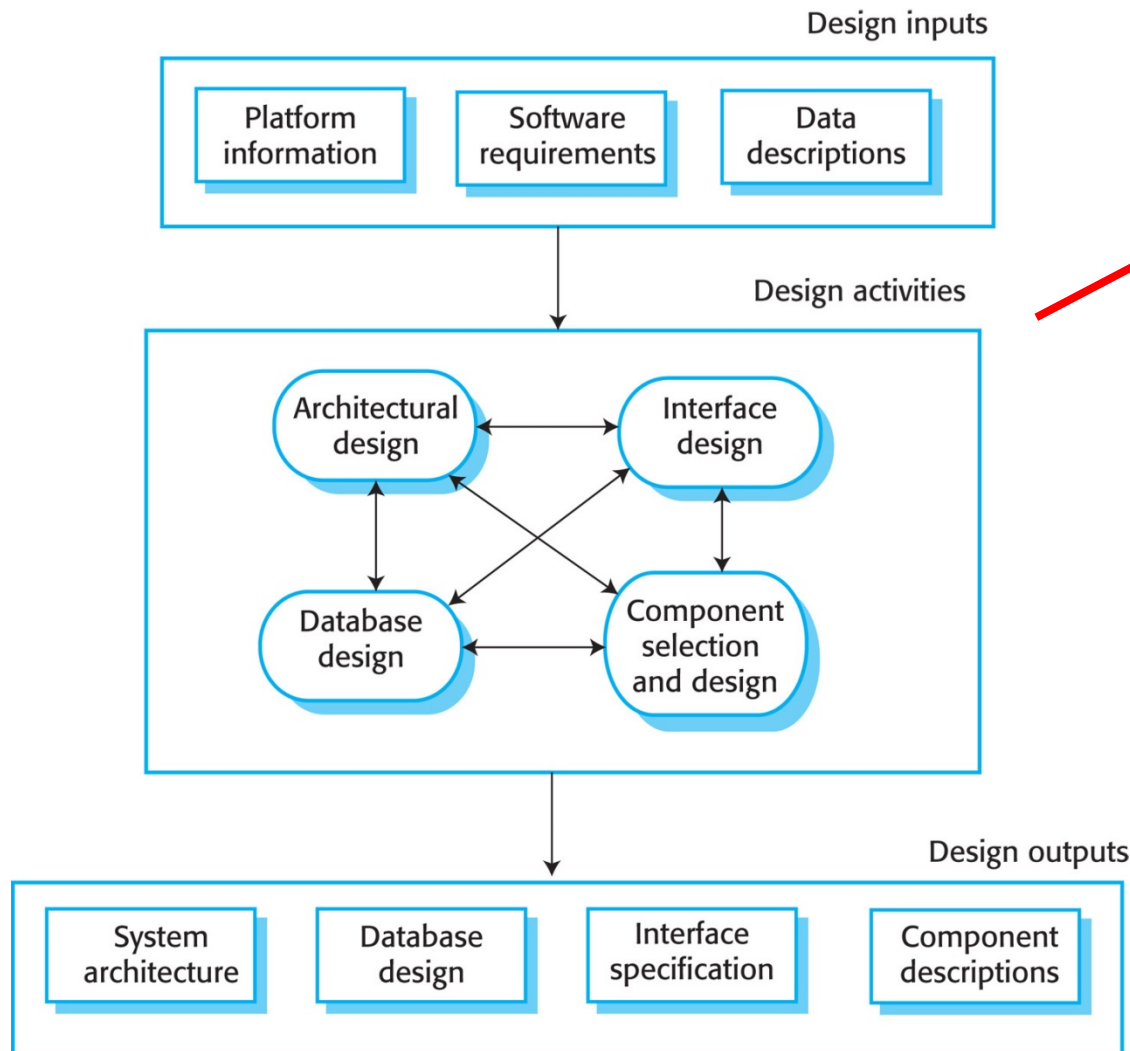
Define the interfaces between system components.

A General Model of the Design Process



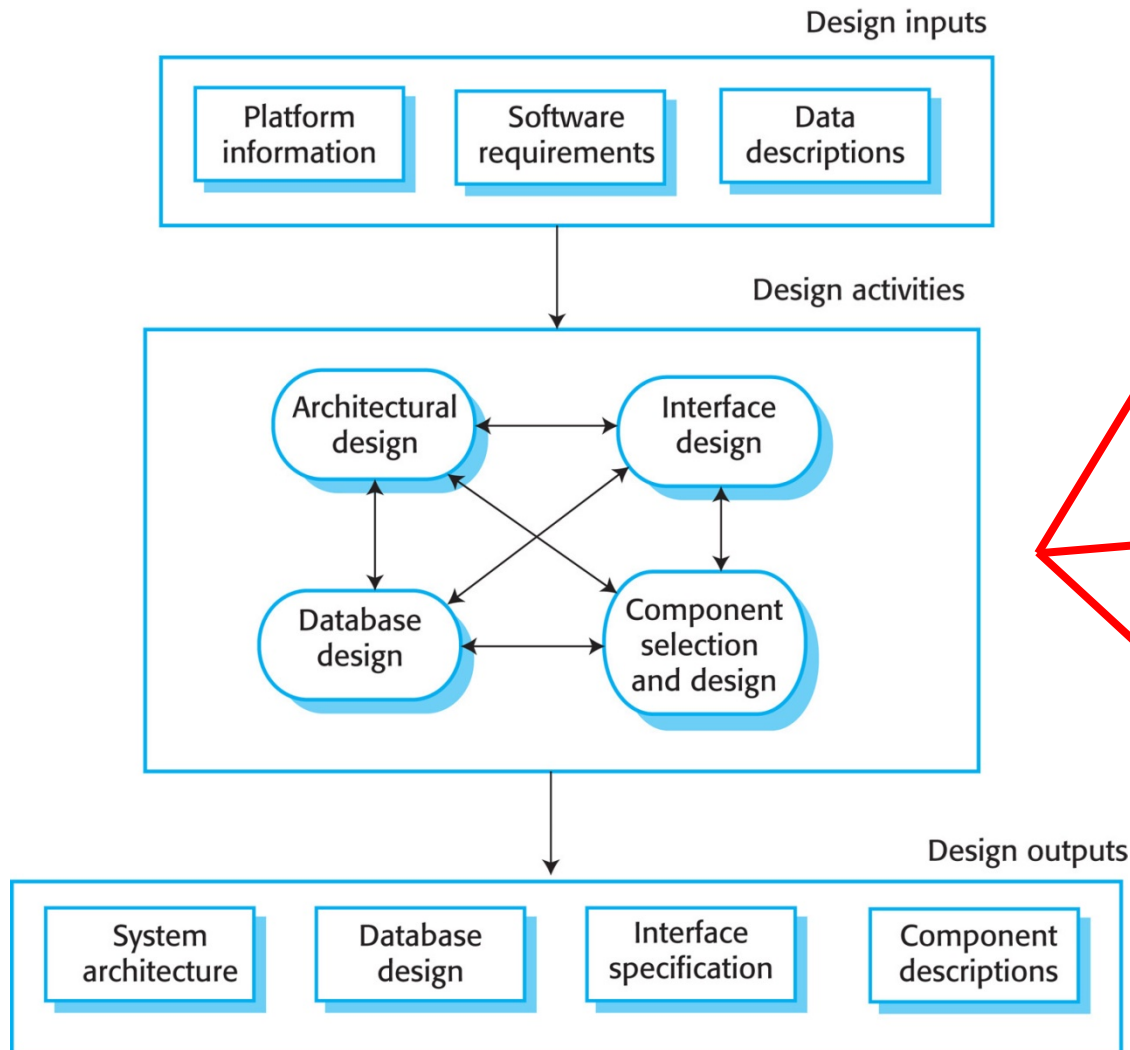
With a precise interface, a component can be used without other components having to know how it is implemented.

A General Model of the Design Process



Once interface specifications are agreed upon, the components can be designed and developed concurrently.

A General Model of the Design Process

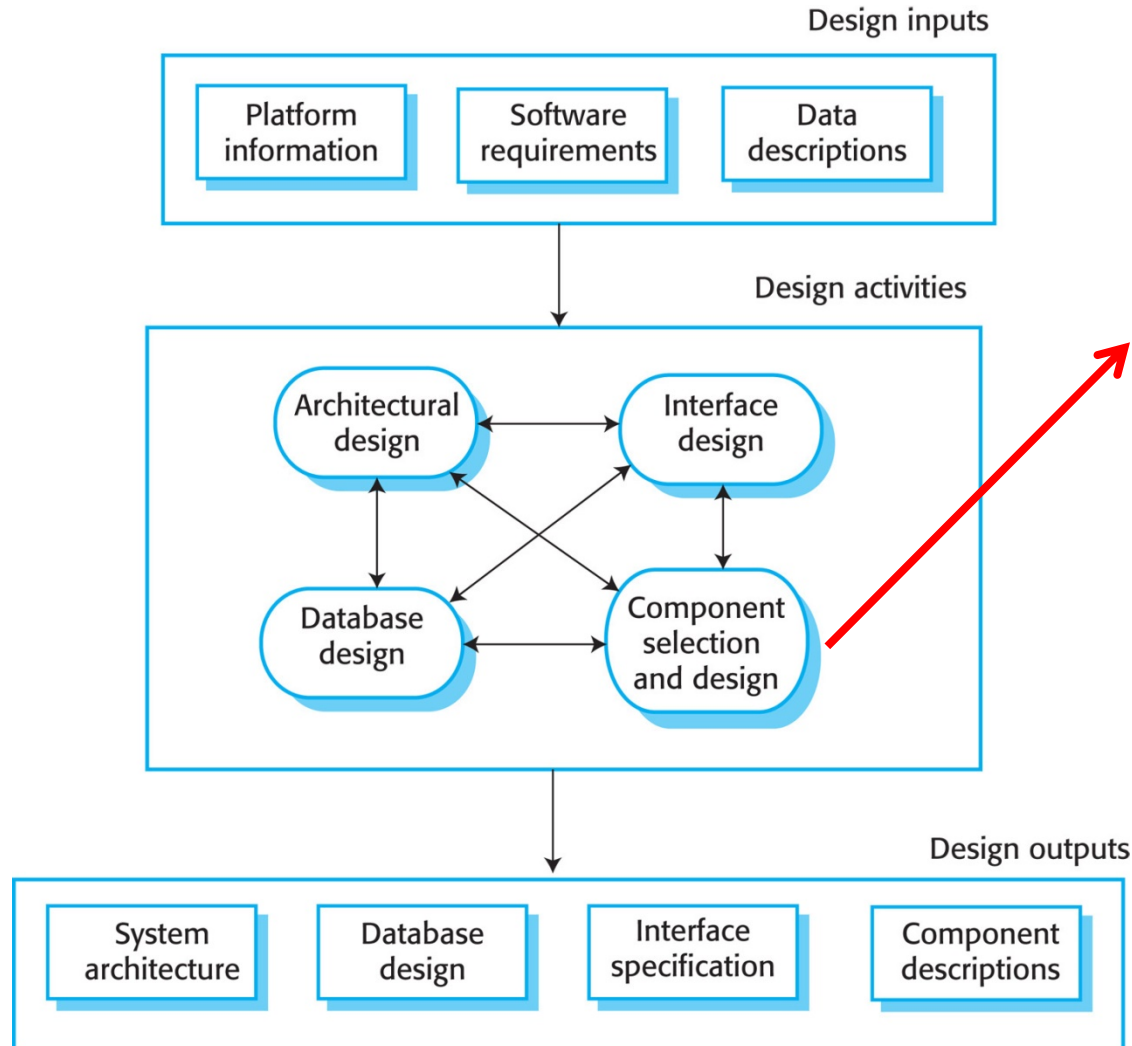


This may be a simple statement of the expected functionality to be implemented, with the specific design left to the programmer.

Or, it may be a list of changes to be made to a reusable component.

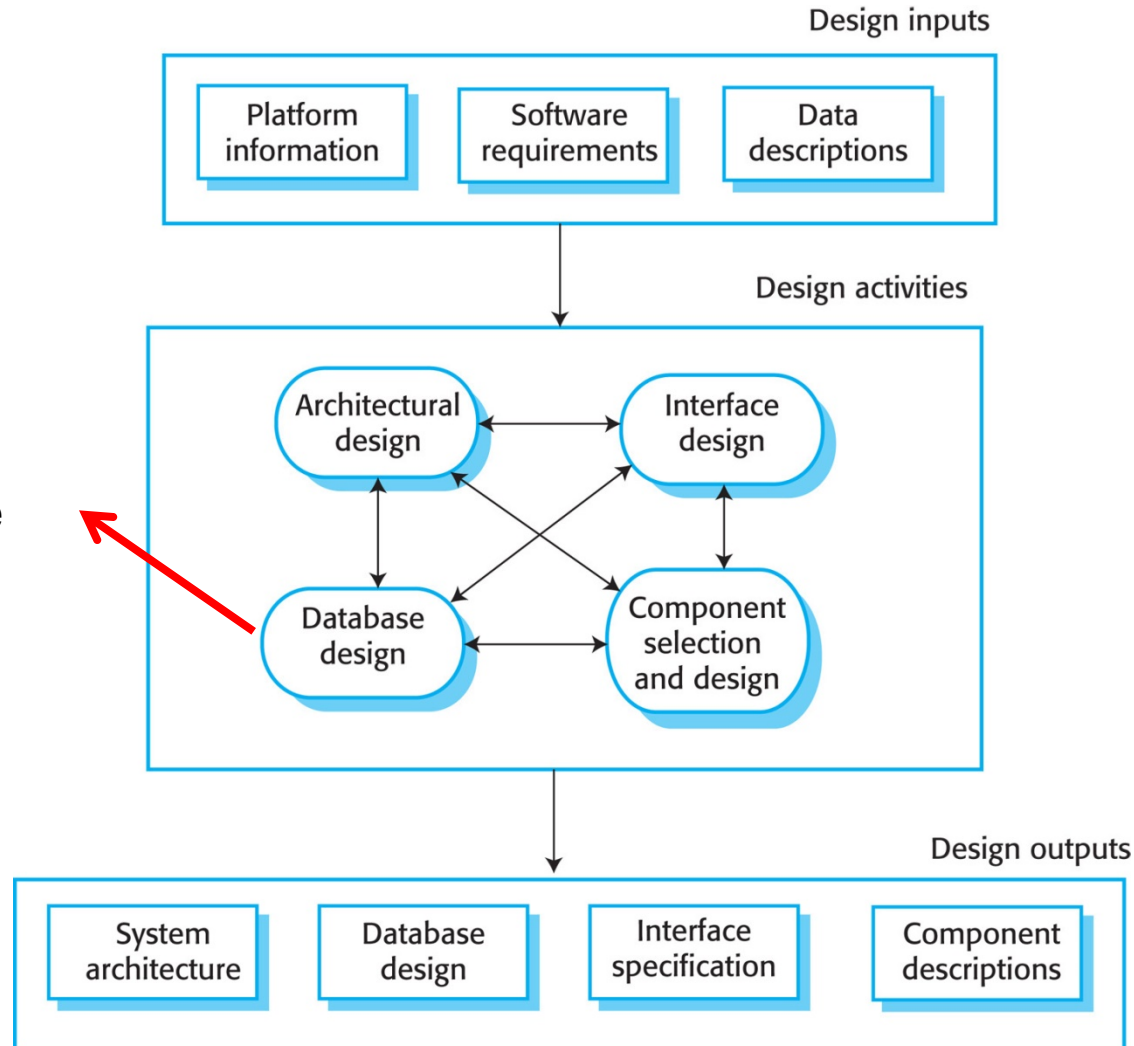
Or, it may be a detailed design model.

A General Model of the Design Process



The design model may be used to automatically generate an implementation (model-driven engineering).

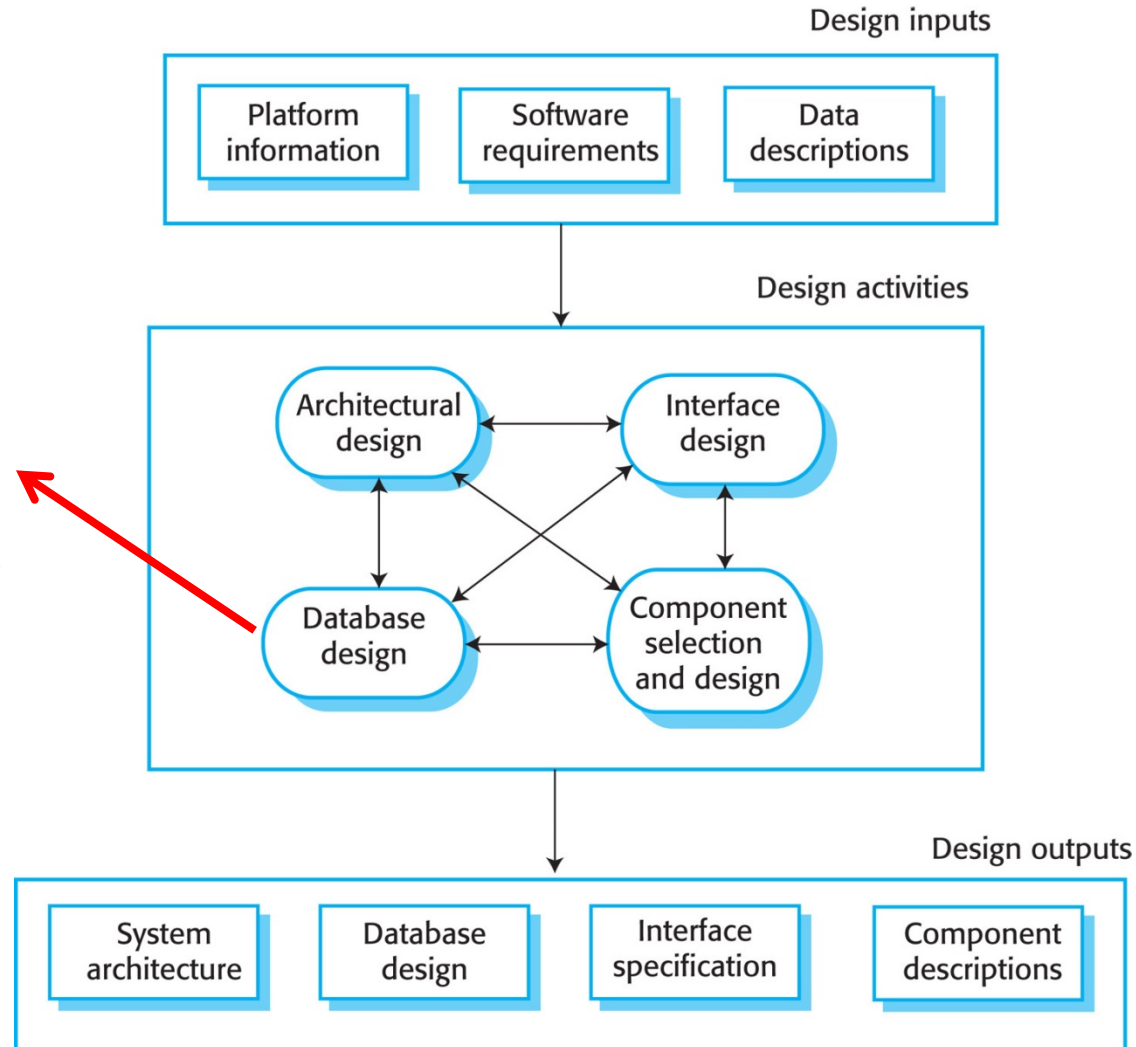
A General Model of the Design Process



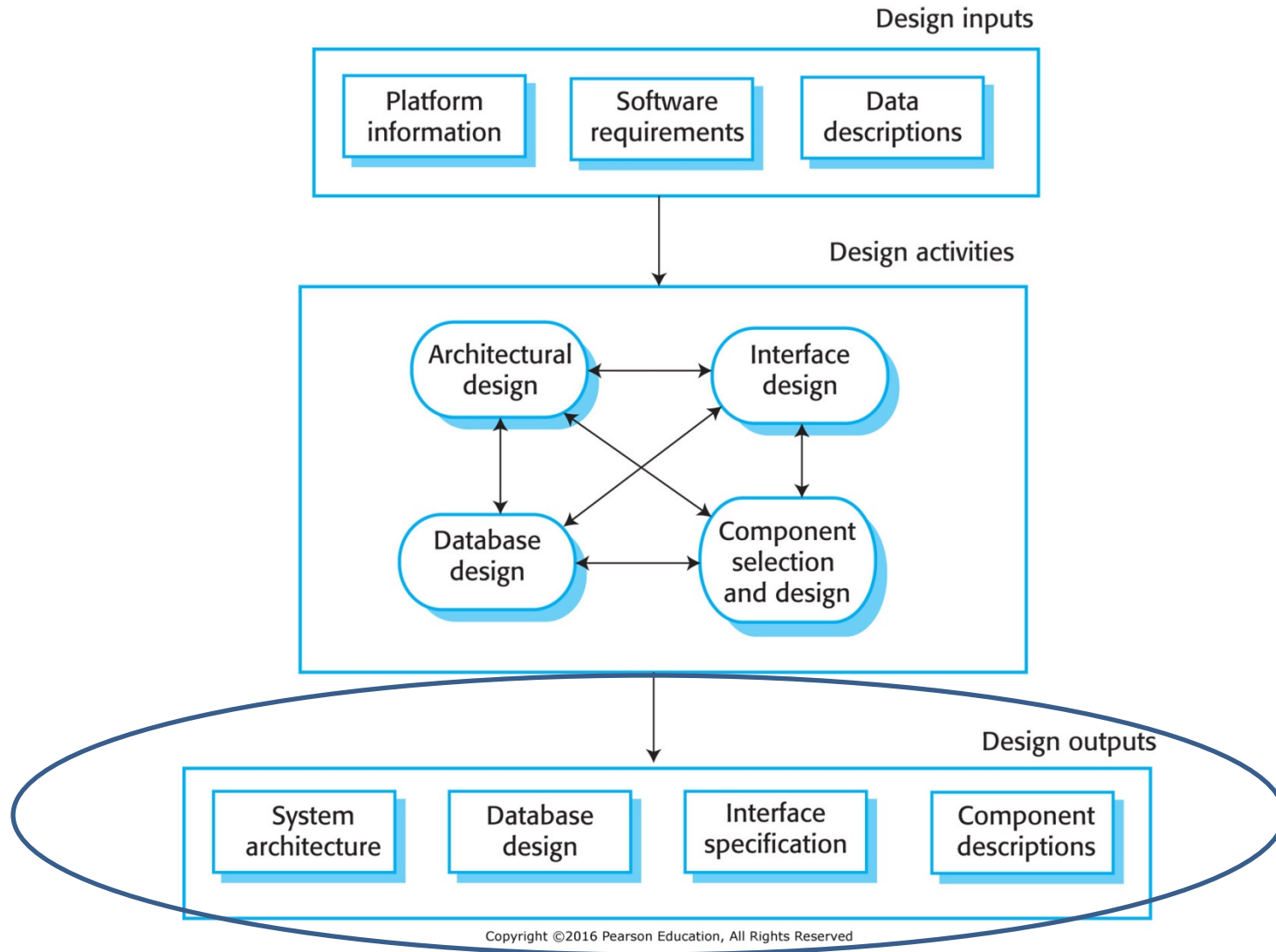
Design the system data structures and how these are to be represented in a database.

A General Model of the Design Process

The work here depends on whether an existing database is to be reused or a new database is to be created.



A General Model of the Design Process



Software Implementation

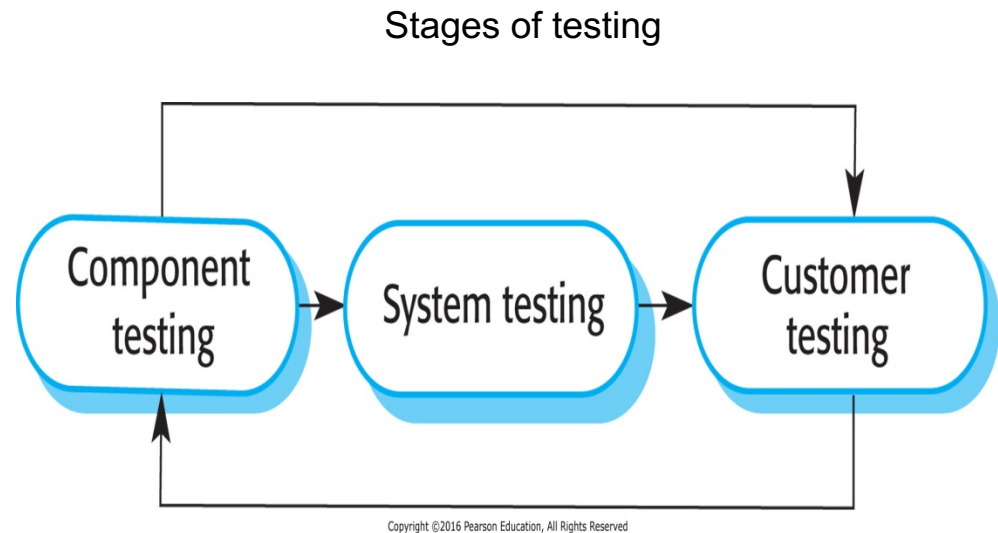
- ❖ Programming is a personal activity.
- ❖ There is no general process to follow.
 - ❖ Some start with components they understand best; some start with those they know the least.
- ❖ (Unit) Testing and debugging are integral parts of software implementation.
 - ❖ Testing and debugging are considered two different processes.
 - ❖ Test cases are usually required for testing.
 - ❖ It's hard enough to find an error in your code when you're looking for it; it's even harder when you've assumed your code is error-free (Steve McConnell, *Code Complete*).

Process Activities (cont.)

- ❖ A software process consists of a sequence of activities that leads to the production of a software product.
- ❖ There are four fundamental activities common to all software processes.
 - ❖ **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation
 - ❖ **Software development**, where the software is designed and programmed
 - ❖ **Software validation**, where the software is checked to ensure that it is what the customer requires
 - ❖ **Software evolution**, where the software is modified to reflect changing customer and market requirements
- ❖ In the waterfall model, they are organized in sequence.
- ❖ In incremental development they are interleaved.

Software Validation

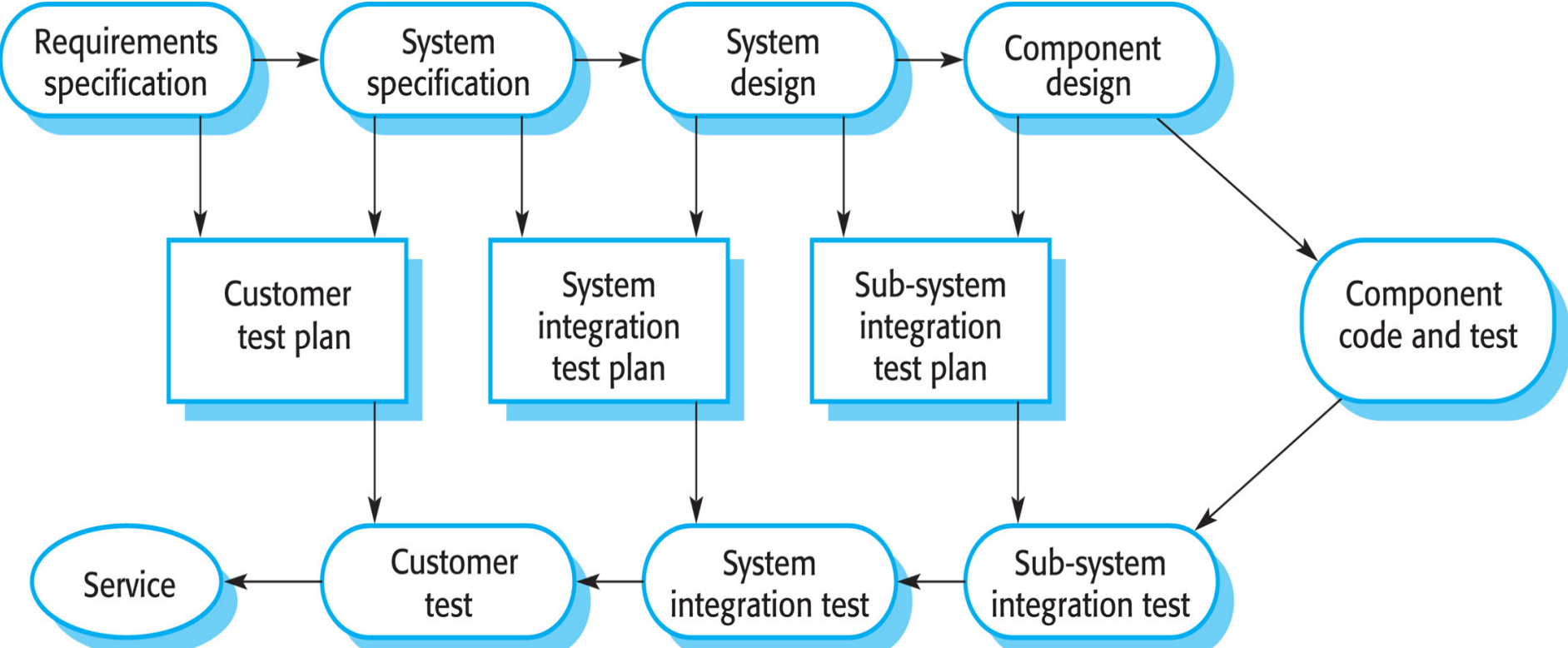
- ❖ Verification and validation (**V & V**) is intended to show that a system conforms to its specification and meets the requirements of the customer.
 - ❖ Involves **review processes** and **system testing**.
 - ❖ System testing involves executing the system with **test cases** that are derived from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



Testing is the most commonly used V & V activity.

Testing Phases

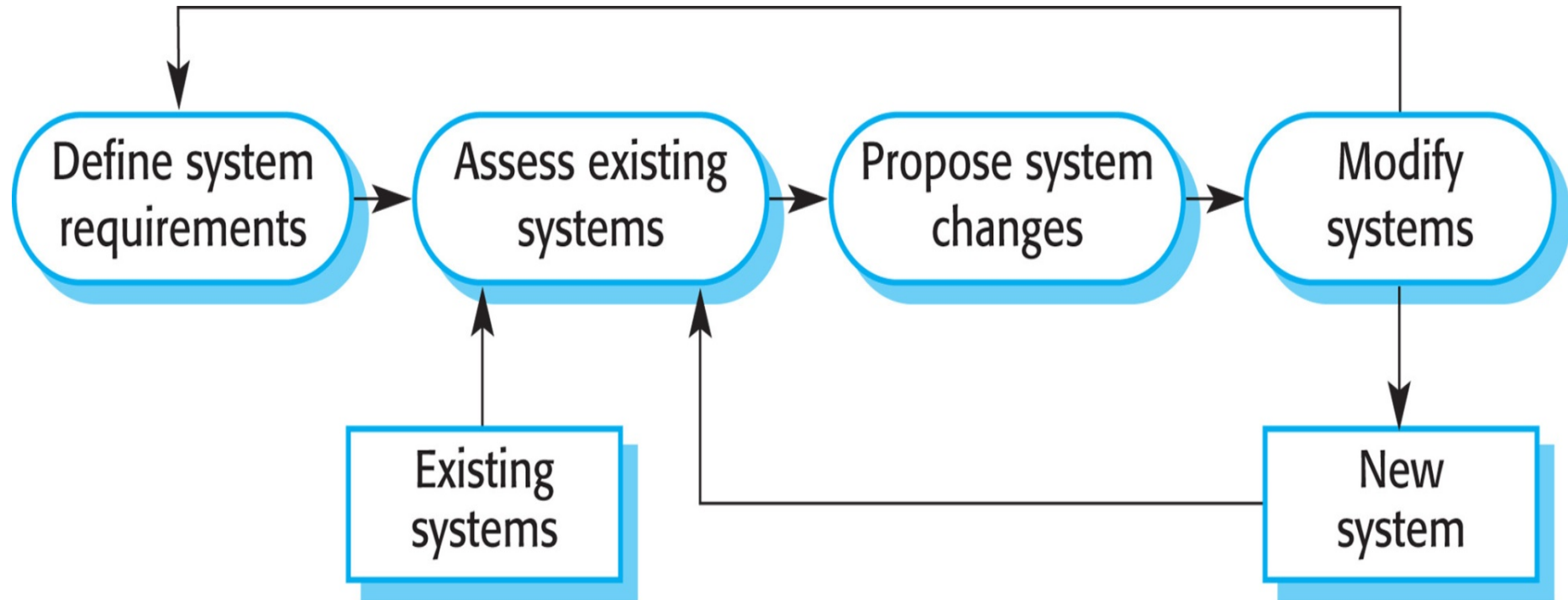
(in a plan-driven software process)



Process Activities (cont.)

- ❖ A software process consists of a sequence of activities that leads to the production of a software product.
- ❖ There are four fundamental activities common to all software processes.
 - ❖ **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation
 - ❖ **Software development**, where the software is designed and programmed
 - ❖ **Software validation**, where the software is checked to ensure that it is what the customer requires
 - ❖ **Software evolution**, where the software is modified to reflect changing customer and market requirements
- ❖ In the waterfall model, they are organized in sequence.
- ❖ In incremental development they are interleaved.

Software Evolution





**SYRACUSE
UNIVERSITY**
**ENGINEERING
& COMPUTER
SCIENCE**

Coping with Change

Week 1: Introduction to Software Engineering and Software Processes

Edmund Yu, PhD
Associate Professor
esyu@syr.edu



**SYRACUSE
UNIVERSITY**
**ENGINEERING
& COMPUTER
SCIENCE**

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., reanalyzing requirements) as well as the costs of implementing new functionality.

Reducing the Costs of Rework

❖ Change avoidance

- ❖ Anticipate possible changes before significant rework is required.
- ❖ Example: (rapid) **prototyping** (prototype development).
- ❖ A prototype may be developed to show some key features of the system.

❖ Change tolerance

- ❖ Structure the software process in such a way that changes can be accommodated at relatively low cost.
- ❖ Example: **incremental delivery**.
- ❖ Proposed changes may be implemented in increments.

Prototype Development

- ❖ A prototype is an initial version of a software system that is used to
 - ❖ Demonstrate concepts
 - ❖ Try out design options
 - ❖ Find out more about the problem and its possible solutions
- ❖ Rapid, iterative development of the prototype is essential so that
 - ❖ Costs are controlled.
 - ❖ Stakeholders can experiment with the prototype early.

Hello World

❖ C:

```
#include <stdio.h>
int main(int argc, char** argv[]) {
    printf("Hello, World!\n");
}
```

❖ Java:

```
public class Hello {
    public static void main(String argv[]) {
        System.out.println("Hello, World!\n");
    }
}
```

❖ Python:

```
print("Hello, World!")
```

❖ Haskell

```
main = putStrLn "Hello, World!"
```

← → W http://en.wikipedia.org/wiki/List_of_graphical_user_interface_builder W List of graphical user interfa... X

prototyping tools

→

→

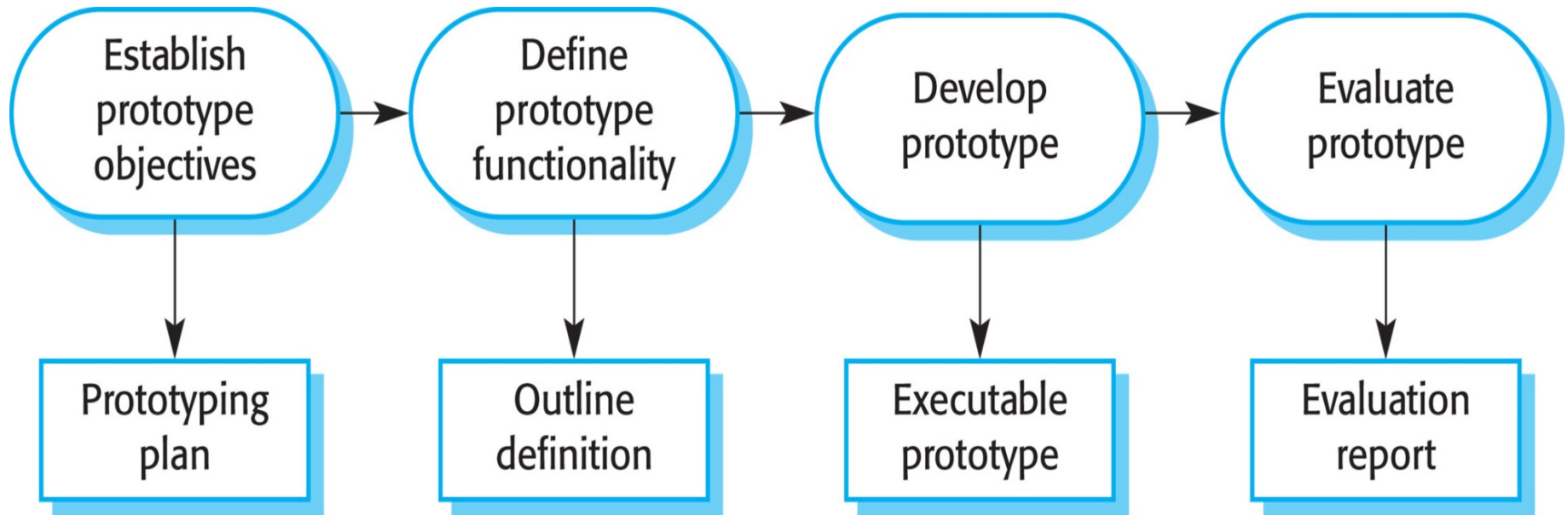
→

- [LANSA](#) is a development environment for generating applications on multiple platforms. One of the main features of LANSA is its high level programming language, called RDML (Rapid Development and Maintenance Language). It is classified as a 4GL (4th generation language). LANSA-developed applications run on many systems including MS Windows, IBM i and Linux.
- [Lazarus](#) is a cross-platform IDE similar to Embarcadero Delphi.
- [LiveCode](#) is drag and drop development environment that runs on OS X, Windows, and Linux, and targets those OSes as well as iOS and Android.
- [m-Power](#) is a Software Development tool which automates application development and rapidly creates enterprise-class Web applications over any database or platform.
- [MyEclipse](#) is a Rapid Application Development environment, focusing on enterprise Java and Web application development. The specialty MyEclipse "Blue Edition" is most similar to [IBM Rational Application Developer](#).
- [NetBeans](#) is a cross-platform, RAD IDE for creating visual desktop, mobile, web, and SOA applications for Linux, Windows and Mac OS X. The IDE officially supports Java, PHP, JavaScript and C/C++ programming languages.
- [nuBuilder](#) is an open source browser based database development tool which stores all forms, reports, data and any custom code in MySQL and displays the content dynamically.
- [Omnis Studio](#) is a cross-platform, Rapid Application Development tool or IDE for creating enterprise and web applications for Windows, Linux, Solaris, and Mac OS X.
- [OpenROAD](#) is a cross-platform IDE for Linux/Unix, Windows with embedded SQL support
- [Panther](#) is a cross-platform ([Windows](#), [Unix](#), [Linux](#); [TUI](#), [GUI](#), [Web](#)), cross-database RAD toolset for development of [client-server](#) and [n-tier](#) database-oriented applications.
- [PureBasic](#) Form Designer is a drag & drop development tool integrated into the [PureBasic](#) IDE that compiles to very compact 32 and 64-bit machine code executables for Windows, Linux and Mac OS X.
- [Xojo](#) (formerly Real Studio) is a cross-platform IDE for creating desktop applications for Windows, Linux and Mac OS X. The language is similar to both VB and Java. It compiles to machine code, uses native controls and produces native executables.
- [RadRails](#) is a cross-platform IDE for creating Ruby on Rails web applications.
- [Servoy](#) Servoy is a cross-platform application development and deployment environment. Servoy consists of a GUI designer, is event-

Prototype Development

- ❖ May be based on rapid prototyping languages or tools (e.g., MyEclipse, NetBeans)
- ❖ 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 nonfunctional requirements such as reliability and security.

Prototype Development



Prototyping Problems

- ❖ A general problem with prototyping is that the prototype may not necessarily be used in the same way as the final system.
 - ❖ The tester of the prototype may not be typical of system users.
 - ❖ The training time during prototype testing may be insufficient.
 - ❖ If the prototype is slow, the testers may adjust their way of working and avoid those system features that have slow response times.
 - ❖ When provided with a better response in the final system, they may use it in a different way.

Reducing the Costs of Rework (cont.)

❖ Change avoidance

- ❖ Anticipate possible changes before significant rework is required.
- ❖ Example: (rapid) **prototyping** (prototype development).
- ❖ A prototype may be developed to show some key features of the system.

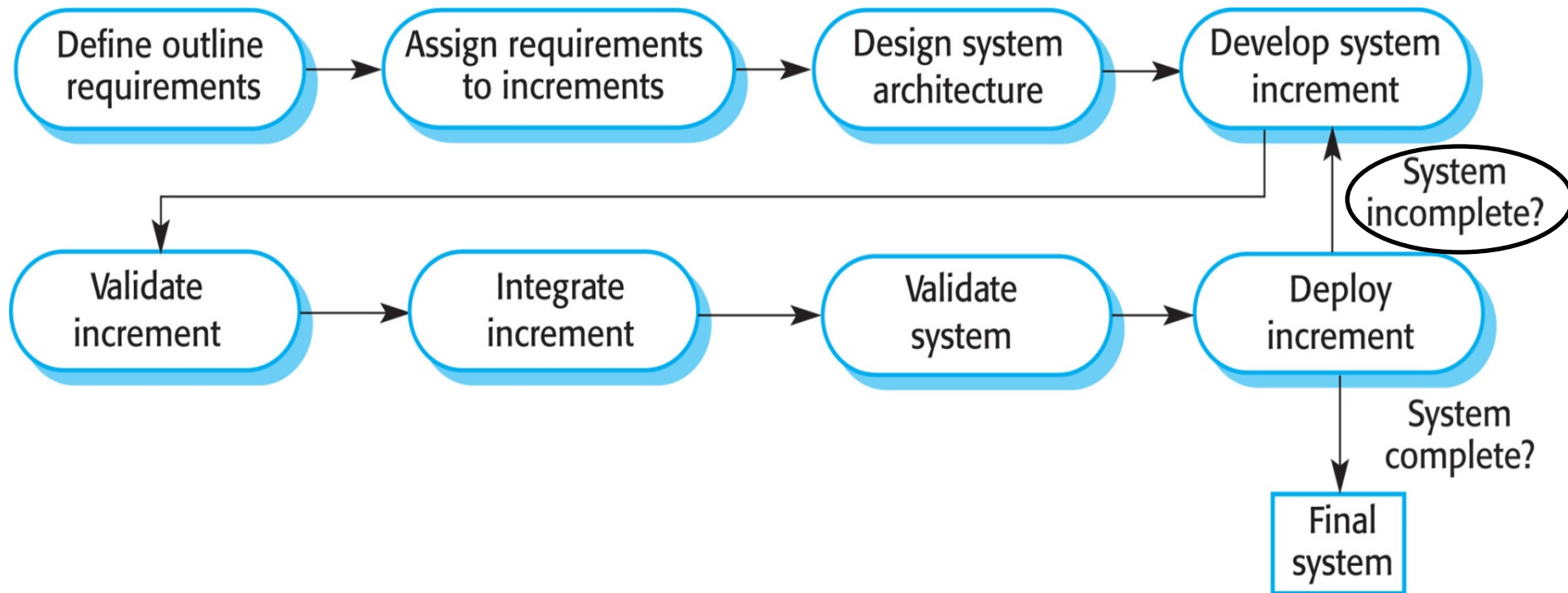
❖ Change tolerance

- ❖ Structure the software process in such a way that changes can be accommodated at relatively low cost.
- ❖ Example: **incremental delivery**.
- ❖ Proposed changes may be implemented in increments.

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 prioritized, and the highest priority requirements are included in early increments.
 - ❖ Once the development of an increment is started, the requirements for that increment are frozen, though requirements for later increments can continue to evolve.

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.
 - ❖ Because 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.
- ❖ Iterative development can also be difficult when a replacement system is being developed.
 - ❖ Users want all of the functionality of the old system and are often unwilling to experiment with an incomplete new system.
- ❖ 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.



SYRACUSE
UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE