

PeopleCert

Software Development Skills

JavaScript Stream

Lesson 6

Study Guide



Copyright Details

The contents of this workshop are protected by copyright and can be reproduced under the Terms of Use agreed between PeopleCert and the ATO using this material only.

Material in this presentation has been sourced from the bibliography listed in the certification's Syllabus. All software-related images are used for educational purposes only and may differ across time.

No part of this document may be reproduced in any form without the written permission of PeopleCert International Ltd. Permission can be requested at www.peoplecert.org.

e-mail: info@peoplecert.org, www.peoplecert.org

Copyright © 2017-2019 PeopleCert International Ltd.

All rights reserved. No part of this publication may be reproduced or transmitted in any form and by any means (electronic, photocopying, recording or otherwise) except as permitted in writing by PeopleCert International Ltd. Enquiries for permission to reproduce, transmit or use for any purpose this material should be directed to the publisher.

DISCLAIMER

This publication is designed to provide helpful information to the reader. Although every care has been taken by PeopleCert International Ltd in the preparation of this publication, no representation or warranty (express or implied) is given by PeopleCert International Ltd. as publisher with respect as to the completeness, accuracy, reliability, suitability or availability of the information contained within it and neither shall PeopleCert International Ltd be responsible or liable for any loss or damage whatsoever (indicatively but not limited to, special, indirect, consequential) arising or resulting of virtue of information, instructions or advice contained within this publication.)

- Document Version: PC-SDS_SG 1.0 | February 2020

PeopleCert

All talents, certified.

PeopleCert: A Global Leader in Certification



- ✓ **Web & Paper based exams in 25 languages**
- ✓ **2,500 Accredited Training Organisations** worldwide
- ✓ **Delivering exams across 200 countries every year**
- ✓ **Comprehensive Portfolio of 500+ Exams and Growing**



PeopleCert

All talents, certified.

© PeopleCert



How to Use This Document

This document is your **PeopleCert Software Developer Skills Study Guide** to help you prepare for the **PeopleCert Software Developer Skills Foundation & Advanced examination**.

It is meant to provide you with a clear outline of everything covered in the course presentation by your instructor that will be on the PeopleCert Software Developer Skills Foundation & Advanced exams.

Your exams will be closed book. You will be given 120 minutes to complete it. It contains 100 multiple choice questions and to pass the exam you must achieve a grade of 65% or higher, or a minimum of 65/100 correct responses. For further details on your exam, including more information on question types and learning objectives, please refer to your course syllabus.

As you follow along, you may see that some material here is not replicated in the trainer presentation. This study guide includes questions, activities, knowledge checks, or other material in the presentation that are facilitated verbally by the instructor. It also does not contain content that is not examinable, but instead is designed to reinforce learning or add value to your course experience. It also provides valuable links and references, throughout the slides, which you can explore further to enhance your learning and understanding of the material provided in the study guide.

PeopleCert

All talents, certified.

Coding Bootcamp

LESSON 6

General Information on the Program

Objectives:

- Introductory Concepts
- Mathematical Computations

Syllabus Items:

- None



Curriculum & Syllabus (Part I)

Topic
Introduction to Software Design and Development
Introduction to Object Oriented Programming
Object Oriented Programming
Relational Databases
Web Design and Development Fundamentals (Front End)
Web Application Development
Web Application Development, MVC and other Frameworks (Basics)
Individual Project (Windows application)

Syllabus (Exam Material)

A. Part I – PeopleCert Developer Skills, Foundation Level

Exam in Java or C# or JavaScript or Python, depending on your selected stream



Curriculum & Syllabus (Part II)

Topic
Web Application Development, MVC and other Frameworks (Advanced)
Web Design and Development Advanced Topics
Object Oriented Programming – Advanced Topics
Advanced Application Development and Other Topics
Testing & Debugging - Advanced
Maintainable Code
Developer Soft Skills, Team Work and Project Documentation
Team/Group Project (Web application)

Syllabus (Exam Material)

B. Part II – PeopleCert Developer Skills, Advanced Level

Exam in Java or C# or JavaScript or Python, depending on your selected stream



Contents | Learning Objectives

- ✓ Acquired the appropriate background to code and extend their programming skills
- ✓ Developed a project portfolio containing at least three full-stack applications and various exercises
- ✓ Learnt to use the latest tools and concepts in application development
- ✓ Worked in a team to design and develop a larger-scale application

Building on the **acquired knowledge**, participants will be able to:

- ✓ Develop full-stack applications using the C# or Java or JavaScript or Python programming languages
- ✓ Start working as full-stack entry-level application developers

In other words...

- ✓ Learn how to **guess**
- ✓ Learn how to **search**



Program Structure

Basic Coding Skills (Foundation Level)

- **7 weeks** – PeopleCert Software Developer Skills, Foundation Level Certificate

Additional Coding Skills (Advanced Level)

- **5 weeks** – PeopleCert Software Developer Skills, Advanced Level Certificate

Technologies

Java	C#	Common
Java	C# programming	HTML / CSS
Maven/Tomcat	ASP .NET MVC	SQL
MySQL / HSDLDB	SQL Server	JavaScript
Servlets / JSP	SQL Reporting Services	Bootstrap Framework
Spring MVC/ Spring Boot / DI / AOP	ADO .NET / Entity Framework	AngularJS / React
JPA / Hibernate	Visual Studio	Version Control (Git)
	Dapper	REST architecture
		Web servers (Apache, nginx)
JavaScript	Python	
JavaScript Programming	Python Programming	xUnit
node.js	Flask	Unix command-line
MySQL	MySQL	MongoDB and other NoSQL databases
Express JS	Django	UI / UX Design
Sails	ActiveRecord ORM	Cloud-based services and their development, like (AWS, Google Cloud Platform)
Waterline		



Coding Bootcamp

Introductory Concepts

- Binary and Other Systems
- Regular Expressions



Coding Bootcamp

Introductory Concepts:

- Binary and Other Systems
- Regular Expressions

Fundamental Questions:

What are the capabilities and limitations of computers and computer programs?

- What can we do with computers/programs?
- Are there things we cannot do with computers/programs?

Studying the Theme:

- How do we prove something **CAN** be done by **SOME** program?
- How do we prove something **CANNOT** be done by **ANY** program?

Languages:

The terms *language* and *word* are used in a strict technical sense in this course:

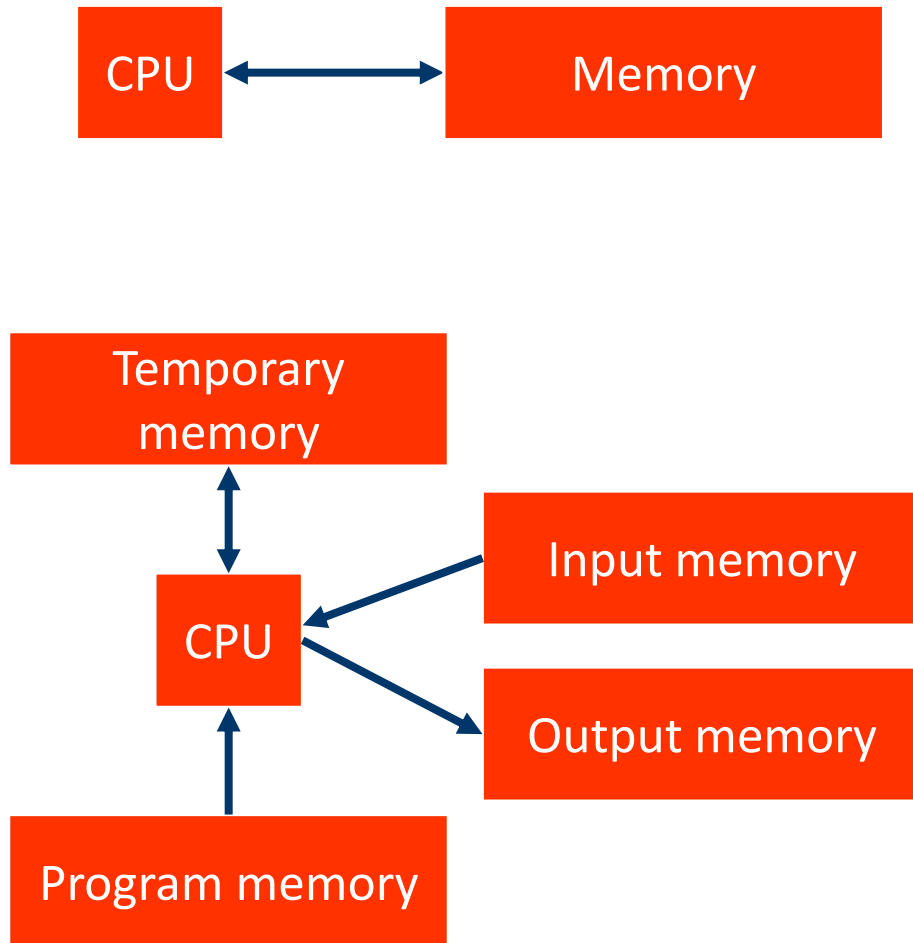
- A *language* is a set of words.
- A *word* is a sequence (or string) of symbols.

Computation

Definition:

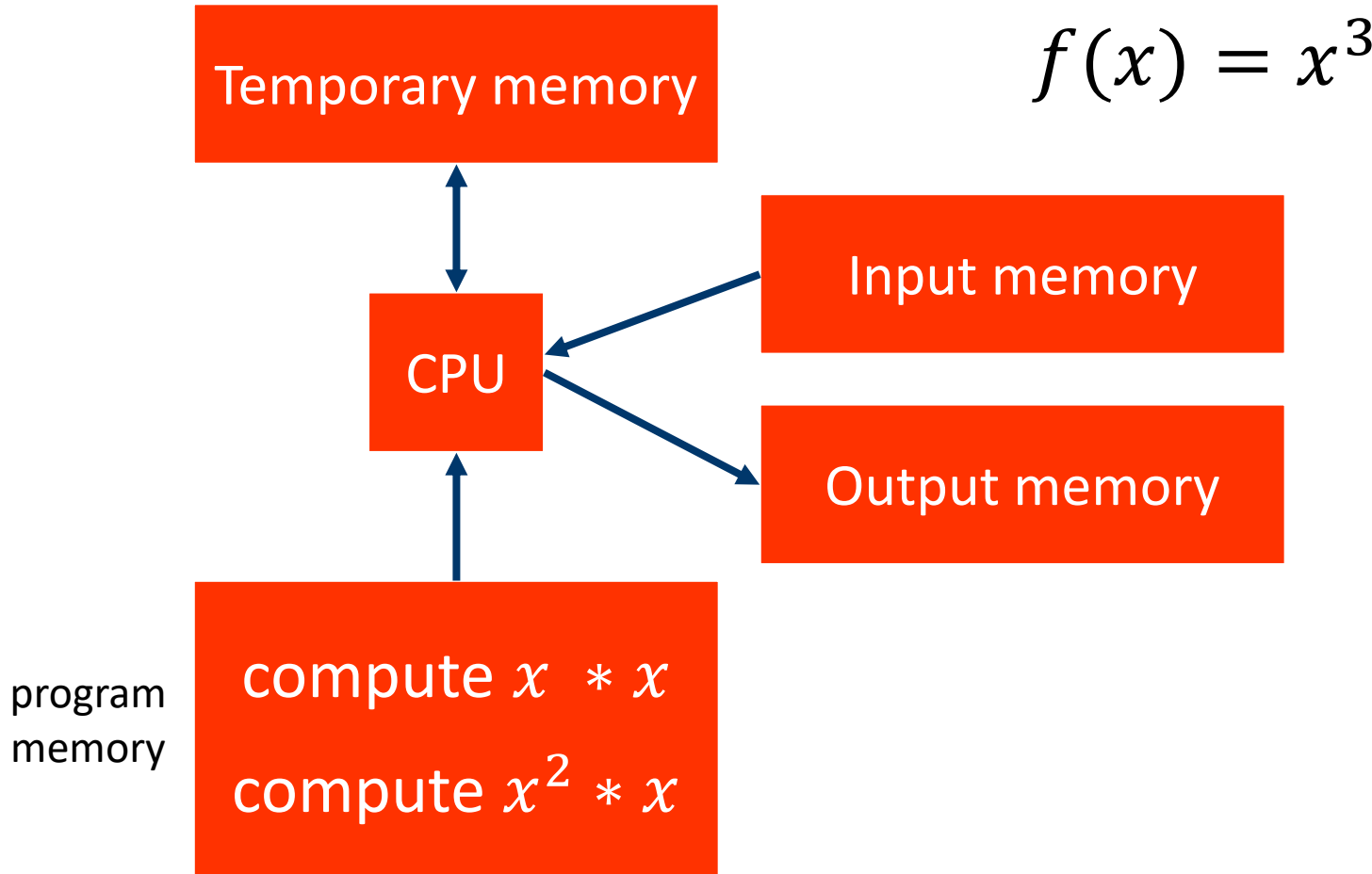
Computation is any type of calculation that includes both arithmetical and non-arithmetical steps and follows a well-defined model, for example an algorithm. The study of computation is paramount (hypernymous) to the discipline of computer science.

Source: <https://en.wikipedia.org/wiki/Computation>



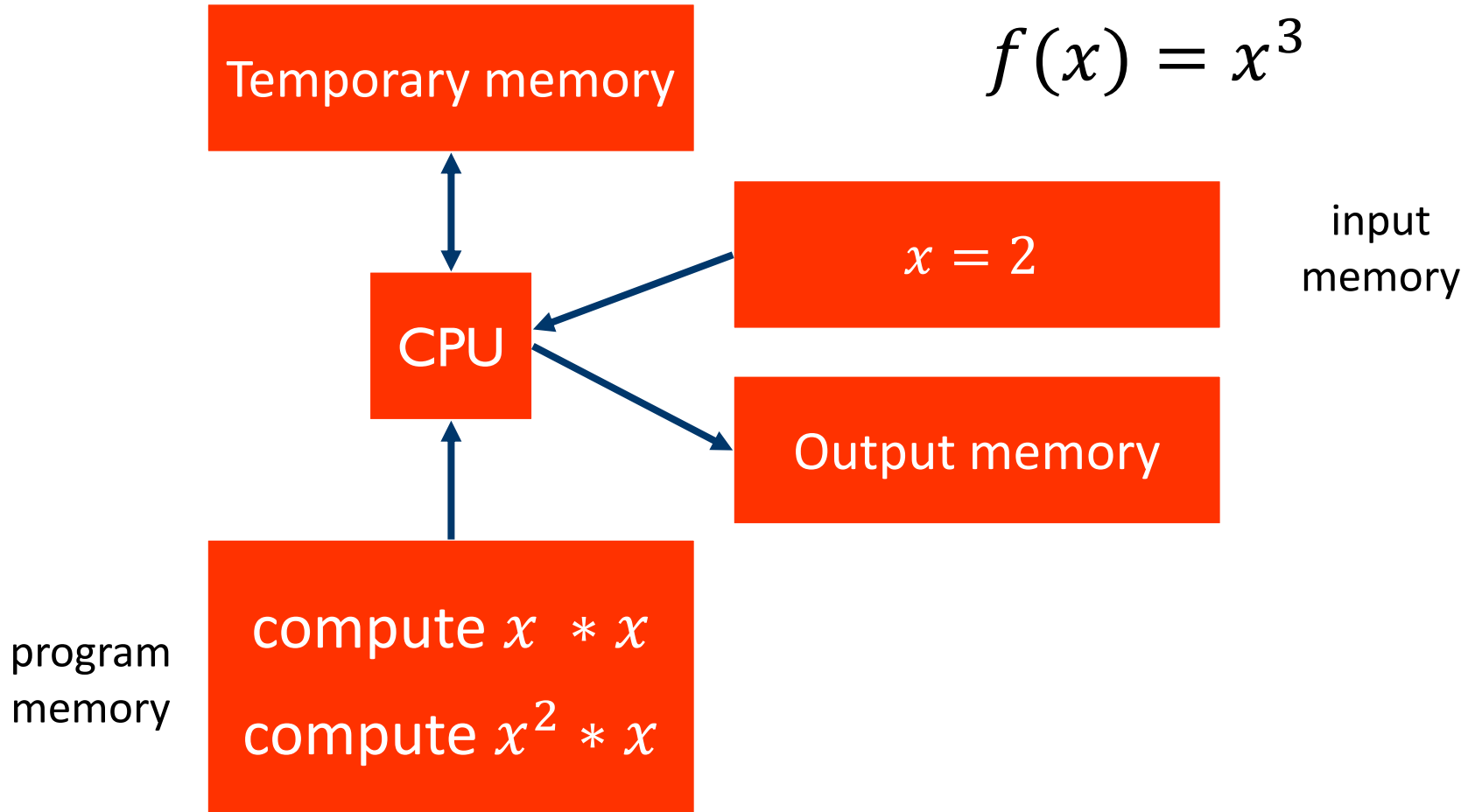


Example (1/4)





Example (2/4)





Example (3/4)

temporary
memory

$$z = 2 * 2 = 4$$
$$f(x) = z * 2 = 8$$

$$f(x) = x^3$$

input
memory

$$x = 2$$

CPU

Output memory

program
memory

compute $x * x$
compute $x^2 * x$



Example (4/4)

temporary
memory

$$z = 2 * 2 = 4$$
$$f(x) = z * 2 = 8$$

$$f(x) = x^3$$

CPU

$$x = 2$$

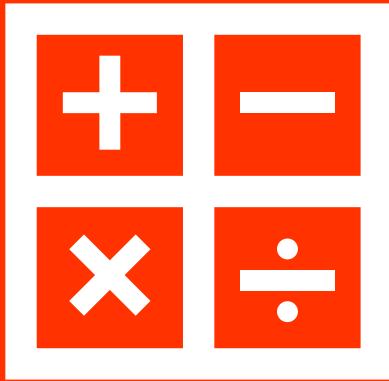
input
memory

$$f(x) = 8$$

output
memory

program
memory

compute $x * x$
compute $x^2 * x$



Coding Bootcamp

Introductory Concepts

- Mathematical Preliminaries



Boolean Operators

Definition:

Boolean Operators are simple words (AND, OR, NOT or AND NOT) used as conjunctions to combine or exclude keywords in a search, resulting in more focused and productive results. This should save time and effort by eliminating inappropriate hits that must be scanned before discarding.

- A Boolean expression is either **true** or **false**
- Using these operators can greatly reduce or expand the amount of records returned.
- Boolean operators are useful in saving time by focusing searches for more 'on-target' results that are more appropriate to your needs, eliminating unsuitable or inappropriate.
- Each search engine or database collection uses Boolean operators in a slightly different way or may require the operator be typed in capitals or have special punctuation.
- **AND**—requires both terms to be in each item returned. If one term is contained in the document and the other is not, the item is not included in the resulting list. (Narrows the search)
- **OR**—either term (or both) will be in the returned document. (Broadens the search)
- **NOT** or **AND NOT** (dependent upon the coding of the database's search engine)—the first term is searched, then any records containing the term after the operators are subtracted from the results. (Be careful with use as the attempt to narrow the search may be too exclusive and eliminate good records). If you need to search the word not, that can usually be done by placing double quotes (<< >>) around it.

Source: <https://library.alliant.edu/screens/boolean.pdf>



Boolean Operators

P	q	!p	p && q	p q	p xor q	p == q
false	false	true	false	false	false	true
false	true	true	false	true	true	false
true	false	false	false	true	true	false
true	true	false	true	true	false	true

<https://library.alliant.edu/screens/boolean.pdf>

One and Zero as True and False

P	q	!p	p && q	p q	p xor q	p == q
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Implication

- A less common Boolean operator **is** \Rightarrow **a**
- The first operation, $x \rightarrow y$, or Cxy , is called **material implication**.
- If x is **true** then the value of $x \rightarrow y$ is taken to be that of y (e.g. if x is true and y is false, then $x \rightarrow y$ is also false).
- But if x is **false** then the value of y can be ignored; however the operation must return *some* Boolean value and there are only two choices. So by definition, $x \rightarrow y$ is *true* when x is false.

P	q	$p \Rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

- It is read as *p implies q*
- q is **true** whenever p is **true**

Source: https://en.wikipedia.org/wiki/Boolean_algebra



Data Representation

Definition:

Data Representation refers to the form in which **data** is stored, processed, and transmitted.

How do computers represent data?

- Most computers are **digital**
 - Recognize only two discrete states: **on** or **off**
 - Use a **binary system** to recognize two states
 - Use number system with two unique digits: 0 and 1, called **bits** (short for **binary digits**)
 - Smallest unit of data computer can process

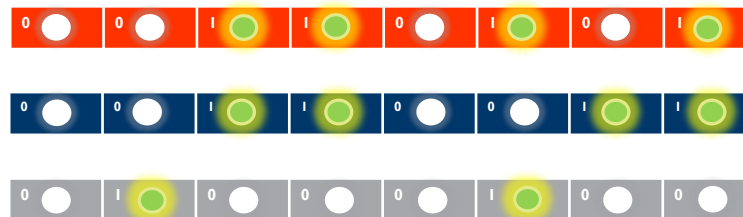
Binary Digit (BIT)	Electronic Charge	Electronic State
1		ON
0		OFF

What is a **byte**?

Definition:

Eight bits grouped together as a unit

- Provides enough different combinations of 0s and 1s to represent 256 individual characters
 - Numbers
 - Uppercase and lowercase letters
 - Punctuation marks



Source: <https://home.adelphi.edu/~siegfried/cs170/170l1.pdf>

Converting Binary to Decimal

Definition

A **decimal number system** relates to, uses, or denotes a system of numerical notation that has 10 as a base.

- Uses **10** numbers: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

23,625					
Power of 10 representation	10^4	10^3	10^2	10^1	10^0
Decimal representation	10000	1000	100	10	1
Base 10 representation	20,000	3,000	600	20	5

Definition

A **binary number system** relates to, uses, or denotes a system of numerical notation that has 2 rather than 10 as a base.

- Uses 2 numbers: **0, 1**

10010001 = 145								
Base 2 representation	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal representation	128	64	32	16	8	4	2	1
Base 2 representation	1	0	0	1	0	0	0	1

Sources: https://en.wikipedia.org/wiki/Binary_number
<https://en.wikipedia.org/wiki/Decimal>

Converting Decimal to Binary

- **Convert decimal 35 to binary**

1. Using 8 bits, find largest power of 2 that will “fit” into 35
2. Place a 1 into that slot
3. If the # doesn’t fit, place a 0 into that slot

Base 2 representation	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal representation	128	64	32	16	8	4	2	1
Base 2 representation	0	0	1	0	0	0	1	1
35 = 00100011								

- **Choose an 8 bit binary number: 10101110**

1. Write the binary digits under the correct column
2. For each column with a 1, you will add that decimal value
3. You will not add the values of the columns you entered 0

Base 2 representation	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal representation	128	64	32	16	8	4	2	1
Base 2 representation	1	0	1	0	1	1	1	0
$128 + 32 + 8 + 4 + 2 = 174$ $10101110 = 174$								

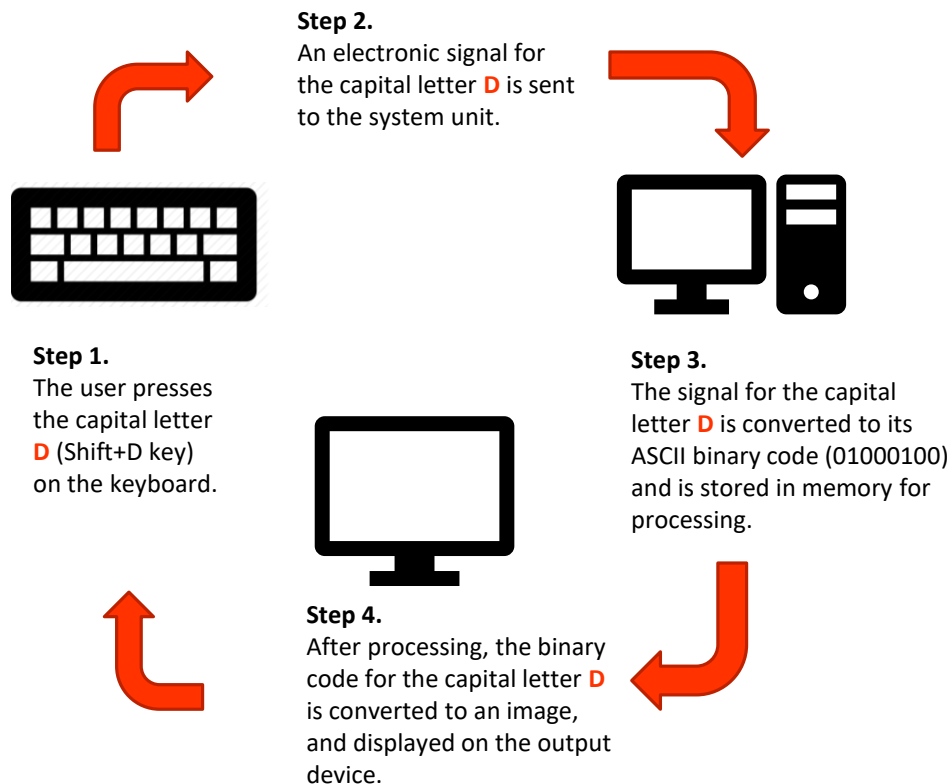
Data Representation

What are **three** popular coding systems to represent data?

- **ASCII** - **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
- **EBCDIC** - **E**xtended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode
- **Unicode** - Coding scheme capable of representing all world's languages

ASCII	Symbol	EBCDIC
00110000	0	11110000
00110001	1	11110001
00110010	2	11110010
00110011	3	11110011

How is a **letter** converted to **binary form** and **back**?



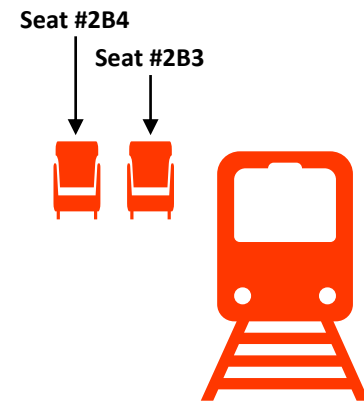
Memory

Definition:

In computing, **memory** refers to a device that is used to store information for immediate use in a computer or related computer hardware device.

What is **memory**?

- Electronic components that store instructions, data, and results
 - Consists of one or more chips on motherboard or other circuit board
 - Each byte stored in unique location called an **address**, similar to addresses on a passenger train
-
- Stores **three basic categories** of items:
 - Operating System and system software
 - Application programs
 - Data and information
 - A **Byte** is the basic storage unit in memory
 - To access data or instructions in memory, computer references the address that contain the bytes of data
 - Manufacturers state the size of memory and storage devices in terms of number of bytes available



Source: https://en.wikipedia.org/wiki/Computer_memory

Memory and Storage Sizes

How is memory **measured**?

- By number of bytes available for storage
- 1 KB = 1024 bytes

Name	Abbreviation	Size
Kilo	K	$2^{10} = 1,024$
Mega	M	$2^{20} = 1,048,576$
Giga	G	$2^{30} = 1,073,741,824$
Tera	T	$2^{40} = 1,099,511,627,776$
Peta	P	$2^{50} = 1,125,899,906,842,624$
Exa	E	$2^{60} = 1,152,921,504,606,846,976$
Zetta	Z	$2^{70} = 1,180,591,620,717,411,303,424$
Yotta	Y	$2^{80} = 1,208,925,819,614,629,174,706,176$

Term	Abbreviation	Approximate Size	Exact Amount	Approximate Number of Pages of Text
Kilobyte	KB or K	1 thousand bytes	1,024 bytes	1/2
Megabyte	MB	1 million bytes	1,048,576 bytes	500
Gigabyte	GB	1 billion bytes	1,073,741,824 bytes	500,000
Terabyte	TB	1 trillion bytes	1,099,511,627,776 bytes	500,000,000

Coding Bootcamp

LESSON 6

Software Design and Development

Objectives:

- Key Computing Concepts
- Software, System Software and Hardware
- Software Architecture
- Software Development Process
- Full-stack Development

Syllabus Items:

- 1.1.1 Define the term computing
- 1.1.2 Define the terms code and program/application
- 1.1.3 Define the terms programming language, compiler and software execution
- 1.1.4 Recognise typical activities in the creation of a program: analysis, design, programming, debugging/testing, maintaining/enhancement
- 1.1.5 Define the term "full-stack development"



Syllabus

Category	Topic	Task
FSD_1 Software Design and Development	1.4 Requirements Capturing and Software Design	1.1.1 Define the term computing
		1.1.2 Define the terms code and program/application
		1.1.3 Define the terms programming language, compiler and software execution
		1.1.4 Recognise typical activities in the creation of a program: analysis, design, programming, debugging/testing, maintaining/enhancement
		1.1.5 Define the term "full-stack development"



Class Exercise | Discussion

- ✓ Know the terms computing, code, program/application, programming language, compiler and software execution
- ✓ Understand the difference between different types of software execution
- ✓ Know the typical activities involved in the creation of a computer program and name all the steps in this process including analysis, design, programming, debugging/testing, maintaining/enhancement
- ✓ Understand the term "full-stack development"



Key Concepts

- **Basic Concepts & Terms**
 - Software
 - Computer program
 - Algorithm
 - System software vs. Application software
 - Compiler and Interpreter
 - Software architecture
 - Full-stack application development



Software

Definition:

Computer software, or simply **software**, is that part of a computer system that consists of encoded information or computer instructions

- Computer hardware and software require each other and neither can be realistically used on its own
- A collection of computer programs, libraries and related data are also referred to as software

Source: <https://www.computerhope.com/jargon/s/software.htm>

Computer program

Definition:

A **computer program** is a collection of instructions that performs a specific task when executed by a computer

- A computer executes the program's instructions in a central processing unit
- A computer program is usually written by a computer programmer in a programming language
- From the program in its human-readable form of source code, a compiler can derive machine code
- Alternatively, a computer program may be executed with the aid of an interpreter

Algorithm

Source: https://en.wikipedia.org/wiki/Computer_program

Definition:

An **algorithm** is a **procedure** or **formula** for solving a problem, based on conducting a sequence of specified actions. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

- A computer program can be viewed as an elaborate algorithm.
- A part of a computer program that performs a well-defined task is known as an algorithm
- An algorithm is a self-contained step-by-step set of operations to be performed
- Algorithms perform calculation, data processing, and/or automated reasoning tasks

Source: <https://whatis.techtarget.com/definition/algorithm>



Class Exercise

Write an **algorithm** for the following problem:

- **Problem:** Given a list of positive numbers, return the largest number on the list.
- **Inputs:** A list L of positive numbers. This list must contain at least one number. (Asking for the largest number in a list of no numbers is not a meaningful question.)
- **Outputs:** A number n , which will be the largest number of the list.



Sample Questions | Answers

Algorithm:

- Set max to 0.
- For each number x in the list L, compare it to max. If x is larger, set max to x.
- max is now set to the largest number in the list.

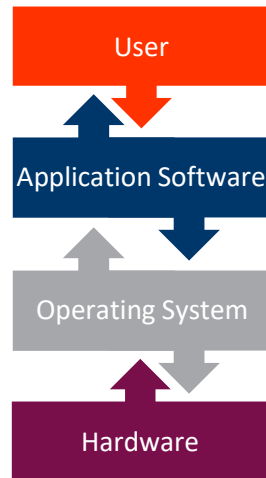
An implementation in Python:

```
function find_max(L) {
    var max = 0;
    var x = 0;
    for(x in L) {
        if(x > max) max = x;
    }
    return max;
}
```

- Does this meet the criteria for being an algorithm?
- *Is it unambiguous?* Yes. Each step of the algorithm consists of primitive operations, and translating each step into Python code is very easy.
- *Does it have defined inputs and outputs?* Yes.
- *Is it guaranteed to terminate?* Yes. The list L is of finite length, so after looking at every element of the list the algorithm will stop.
- *Does it produce the correct result?* Yes. In a formal setting you would provide a careful proof of correctness. In the next section I'll sketch a proof for an alternative solution to this problem.

Source: <http://fiftyexamples.readthedocs.io/en/latest/algorithms.html>

Software, System Software and Hardware



An **operating system (OS)** is **software**, that manages **computer hardware** resources, runs other programs, and provides common services for the **user** and **application software**.

Application Software

Definition:

A computer program designed to perform a group of coordinated functions, tasks, or activities for the benefit of the user

- App or application for short
- The collective noun "**application software**" refers to all applications collectively (contrasts with system software)
- Applications built for mobile platforms are called mobile apps

Examples:

A word processor, a spreadsheet, an accounting application, a web browser etc.

Source: https://en.wikipedia.org/wiki/Application_software

Software Architecture

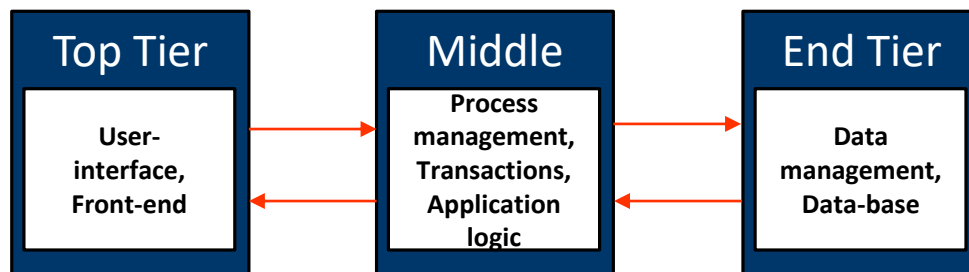
- Software architecture refers to the fundamental structures of a software system
- Each structure comprises software elements, relations among them, and properties of both elements and relations
- The architecture of a software system is a metaphor, analogous to the architecture of a building

Full Stack Development

- Refers to software development covering all the layers of **a 3-tier architecture**:

Definition:

*A **3-tier architecture** is a type of software architecture which is composed of three “tiers” or “layers” of logical computing. They are often used in applications as a specific type of client-server system.*



- **Presentation (Top) Tier:** The presentation tier is the front end layer in the 3-tier system and consists of the user interface.
- **Application (Middle) Tier:** The application tier contains the functional business logic which drives an application's core capabilities (written in Java, .NET, C#, Python, C++, etc.)
- **Data (End) Tier:** The data tier comprises of the database/data storage system and data access layer (MySQL, Oracle, Microsoft SQL Server etc.) Data is accessed by the application layer via API calls.

Source: <https://www.jinfony.com/resources/bi-defined/3-tier-architecture-complete-overview/>



Software Development Process

Core Activities:

- Requirements capturing (also called *Requirements analysis*)
- Design
- Construction (also called *Software implementation*)
- Software Testing and Debugging
- Deployment
- Maintenance

Typical Activities in the Creation of a Program

- Analysis
- Design
- Programming
- Debugging/testing
- Maintaining/enhancement



It is time for a Knowledge Check!



Sample Questions

1. Computing is defined as:
 - A. Any goal-oriented activity requiring, benefiting from, or creating computers
 - B. An algorithm that uniquely represents symbols from some source alphabet, which may be in some other target alphabet
 - C. A deliberate process that transforms one or more inputs into one or more results, with variable change
 - D. A process of discovering and resolving defects that prevent correct operation of computer software or a system
2. Which of the following is a typical activity in the creation of a program?
 - A. Prototyping
 - B. Project Management
 - C. Rapid Application
 - D. Debugging
3. The fundamental structures of a software system are referred to as:
 - A. Computing
 - B. Algorithm
 - C. Software Architecture
 - D. Application Software



Sample Questions | Answers

1. Computing is defined as:
 - A. **Any goal-oriented activity requiring, benefiting from, or creating computers**
 - B. An algorithm that uniquely represents symbols from some source alphabet, which may be in some other target alphabet
 - C. A deliberate process that transforms one or more inputs into one or more results, with variable change
 - D. A process of discovering and resolving defects that prevent correct operation of computer software or a system
2. Which of the following is a typical activity in the creation of a program?
 - A. Prototyping
 - B. Project Management
 - C. Rapid Application
 - D. **Debugging**
3. The fundamental structures of a software system are referred to as:
 - A. Computing
 - B. Algorithm
 - C. **Software Architecture**
 - D. Application Software

Coding Bootcamp

Lesson 6

Software Architectures

Objectives:

- Software Architecture

Syllabus Items:

- 1.2 Software Architectures



Syllabus

Category	Topic	Task
FSD_1 Software Design and Development	1.2 Software Architecture	1.2.1 Understand the notion of software architecture
		1.2.2 Understand the notion of the client-server model
		1.2.3 Understand typical web-application architecture
		1.2.4 Understand three-tier architecture
		1.2.5 Understand the end-to-end workflow of a web request
		1.2.6 Understand the software-as-a-service concept
		1.2.7 Distinguish between centralized versus distributed application architectures: peer-to-peer, microservices etc.
		1.2.8 Understand the architecture of popular applications



Contents | Learning Objectives

- ✓ Understand what Software Architecture is and what are some typical such architectures, as well as be able to understand the architecture used in popular applications
- ✓ Understand the Client-Server, web application and three-tier architectures
- ✓ Understand the software as a service (SaaS) concept and its benefits
- ✓ Know the differences between centralized versus distributed application architectures, including peer-to-peer, microservices etc.
- ✓ Understand mobile application architecture



Software Architecture

Definition:

***Software architecture** refers to the fundamental structures of a software system and the discipline of creating such structures and systems*

Some characteristics of software architecture include:

- The **high level structures** of a software system, the **discipline** of creating such structures, and the **documentation** of these structures
- These structures are needed to **reason** about the software system.
- Each **structure** comprises of:
 - **software** elements,
 - **relations** among them, and
 - **properties** of both elements and relations
- The **architecture** of a software system is a metaphor, analogous to the architecture of a building
- It is vital to understand when to make specific structural options from the available possibilities during the design of software as making fundamental **structural choices** can be **costly** to change once implemented
- **Documenting** software architecture facilitates communication between stakeholders, captures early decisions about the high-level design, and allows reuse of design components between projects

Source: https://en.wikipedia.org/wiki/Software_architecture



Software Architecture (2)

Software architecture exhibits the following:

- **Multitude of stakeholders:** software systems have to cater to a variety of stakeholders such as business managers, owners, users, and operators.
- **Separation of concerns:** the established way for architects to reduce complexity is to separate the concerns that drive the design.
- **Quality-driven:** classic software design approaches were driven by required functionality and the flow of data through the system, but the current insight is that the architecture of a software system is more closely related to its quality attributes such as fault-tolerance, backward compatibility, extensibility, reliability, maintainability, availability, security, usability, and other such –ilities. Stakeholder concerns often translate into requirements on these quality attributes, which are variously called non-functional requirements, extra-functional requirements, behavioral requirements, or quality attribute requirements.
- **Recurring styles:** like building architecture, the software architecture discipline has developed standard ways to address recurring concerns. These "standard ways" are called by various names at various levels of abstraction.
- **Conceptual integrity:** the idea that the architecture of a software system represents an overall vision of what it should do and how it should do it. This vision should be separated from its implementation.
- **Cognitive constraints:** organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations

Source: https://en.wikipedia.org/wiki/Software_architecture



Software Architecture (3)

There are many activities that a software architect performs. A software architect typically works with project managers, discusses architecturally significant requirements with stakeholders, designs a software architecture, evaluates a design, communicates with designers and stakeholders, documents the architectural design and more

- **Four** core activities in software architecture design.
- These core architecture activities are performed **iteratively** and at different stages of the initial software development life-cycle, and over the evolution of a system.
 - **Architectural analysis:** the process of understanding the environment in which a proposed system will operate and determining the requirements for the system. Can include items such as:
 - what the system will do when operational
 - how well the system will perform runtime
 - development-time
 - business requirements and environmental contexts of a system
 - **Architectural synthesis or design:** the process of creating an architecture.
 - **Architecture evaluation:** the process of determining how well the current design or a portion of it satisfies the requirements derived during analysis
 - **Architecture evolution:** the process of maintaining and adapting an existing software architecture to meet changes in requirements and environment.
- Architecture requires **critical supporting activities**. These supporting activities take place throughout the core software architecture process. They include *knowledge management* and *communication*, *design reasoning* and *decision making*, and *documentation*.

Source: https://en.wikipedia.org/wiki/Software_architecture



Discussion

Example:

- The systems that controlled the space shuttle launch vehicle had the requirement of being very fast and very reliable.
- Therefore, an appropriate real-time computing language would need to be chosen.
- Additionally, to satisfy the need for reliability the choice could be made to have multiple redundant and independently produced copies of the program, and to run these copies on independent hardware while cross-checking results.



Software Architecture (4)

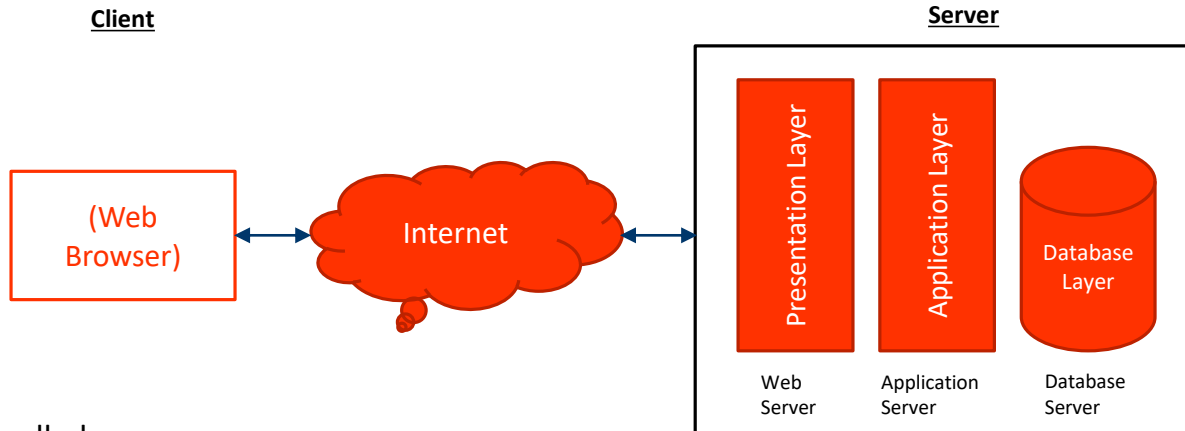
In this lesson, we will discuss:

- Architectural Design
- Detailed Software design
- Client-server architecture
- 3-tier architecture
- Typical web application architecture

Architectural Design

Definition:

IEEE defines **architectural design** as “the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.”



- Also called
 - **High Level Architecture**
 - **High-level view of a system**

Detailed Software Design

- Depicts relationships among software elements/ components at lower level
- **Examples:** UI design, class diagram, database schema, component diagram

Source: <https://www.geeksforgeeks.org/software-engineering-architectural-design/>



Example: User-interface / Mock-up Design

5. Lumzy



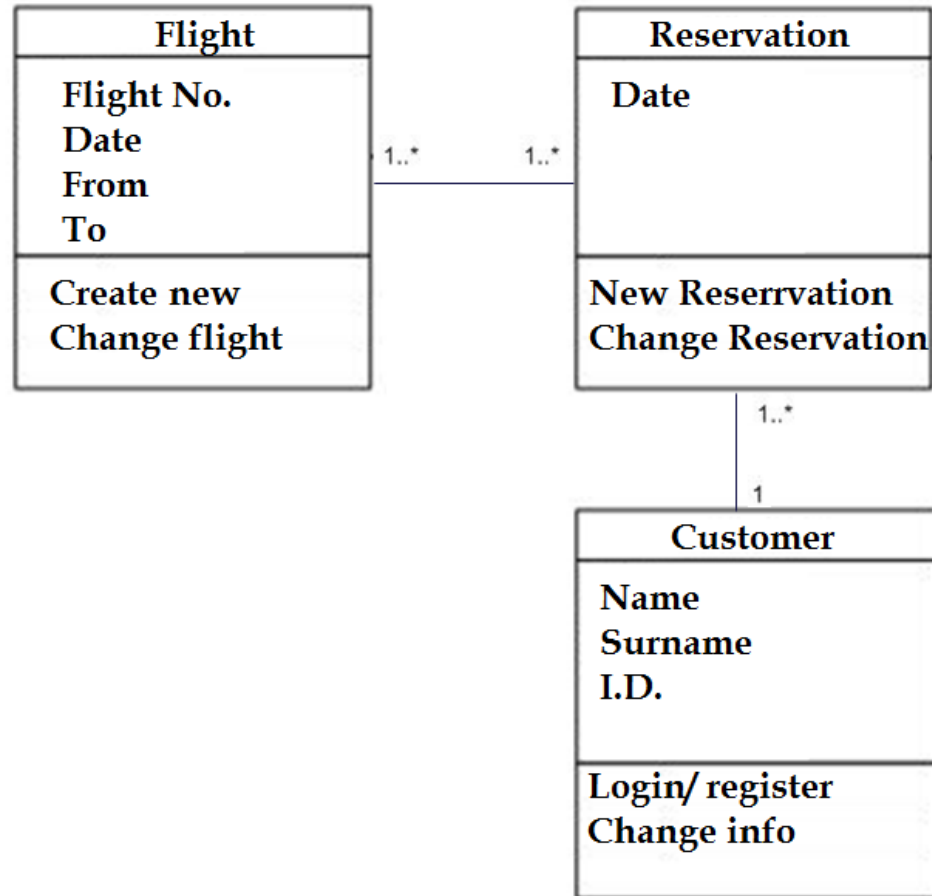
1. Mockingbird



9. Denim

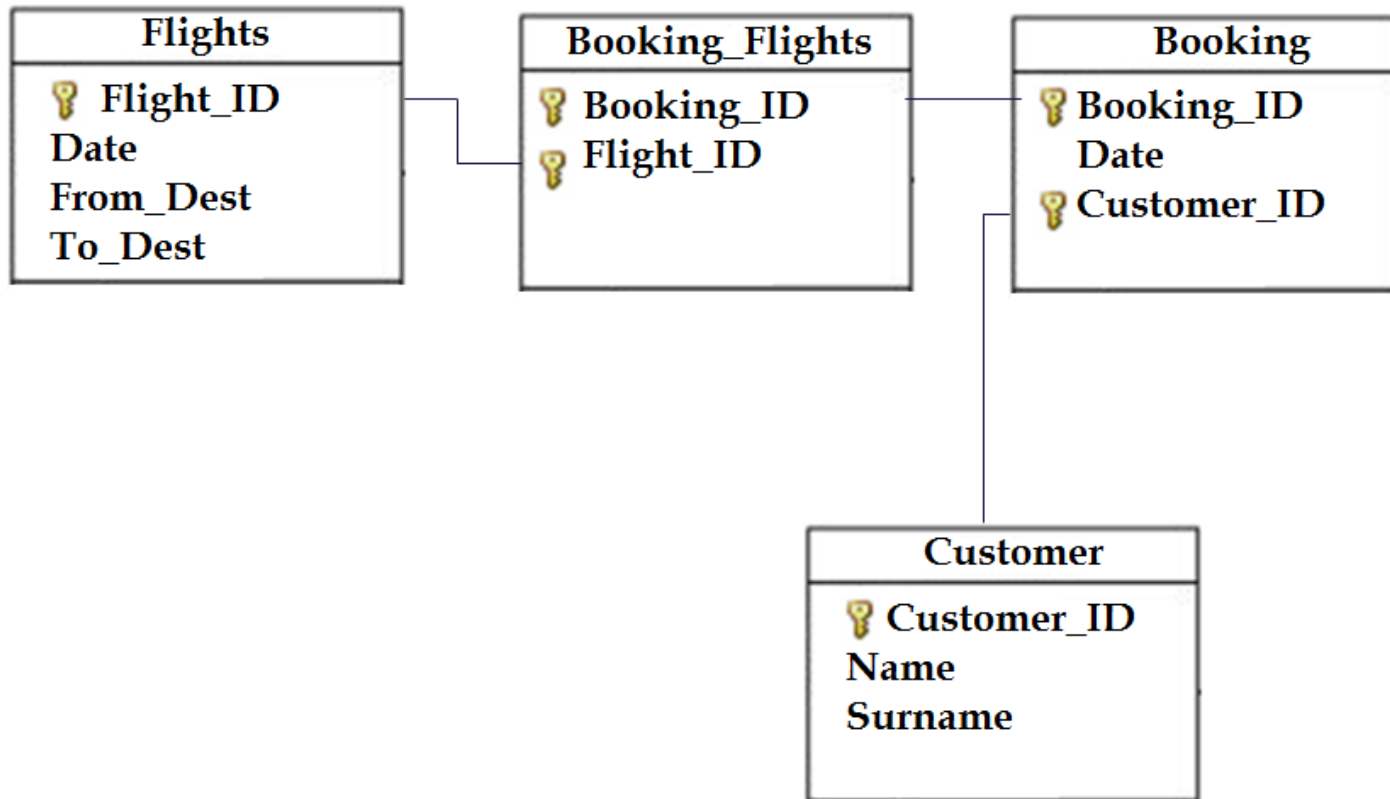


Example: Class Diagram



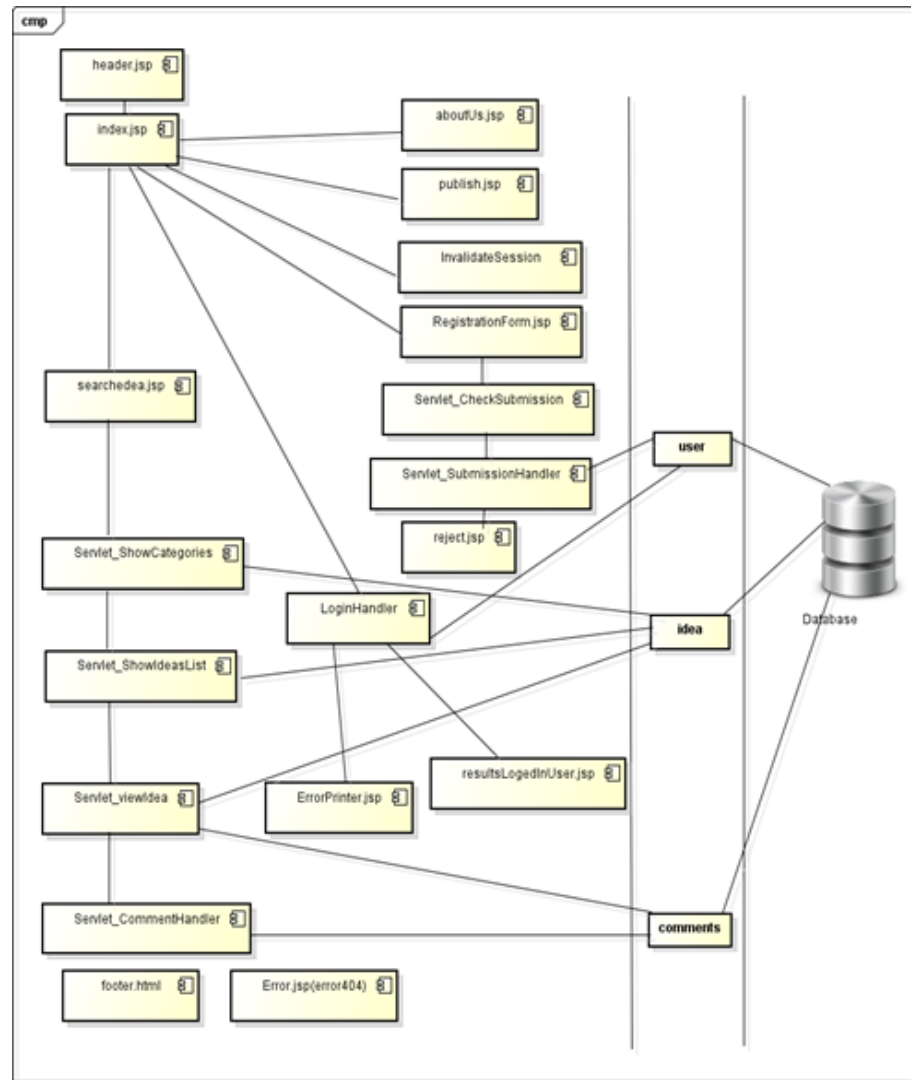
Source: <https://iansommerville.com/software-engineering-book/>

Example: Database Schema



Source: <https://iansommerville.com/software-engineering-book/>

Example: Component Diagram



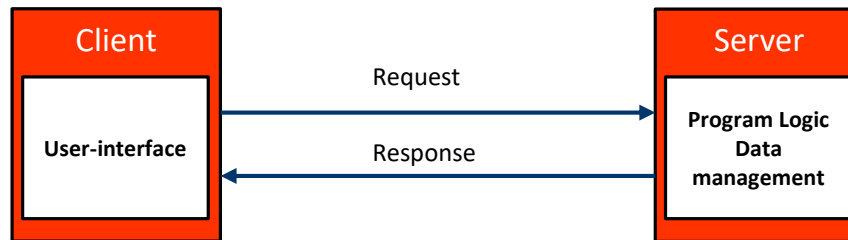
Source: <https://iansommerville.com/software-engineering-book/>

Client-Server Architecture

Definition:

A **Client-server architecture**, refers to the architecture of a computer network in which many clients (remote processors) request and receive service from a centralized server (host computer). Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. Servers wait for requests to arrive from clients and then respond to them.

- **Server (host):** A centralized, host computer
- **Clients** (remote processors) request and receive service from a centralized server (host computer).



- **Client computers** provide an interface to allow a computer user to **request services** of the server and to display the results the server returns. Clients are often situated at workstations or on personal computers, while servers are located elsewhere on the network, usually on more powerful machines.
- **Servers** wait for requests to arrive from clients and then respond to them. Ideally, a server provides a standardized transparent interface to clients so that clients need not be aware of the specifics of the system (i.e., the hardware and software) that is providing the service.
- This computing model is especially effective when clients and the server each have distinct tasks that they routinely perform.
- Many clients can access the server's information simultaneously, and, at the same time, a client computer can perform other tasks, such as sending e-mail. Because both client and server computers are considered intelligent devices, the client-server model is completely different from the old "mainframe" model, in which a centralized mainframe computer performed all the tasks for its associated "dumb" terminals.

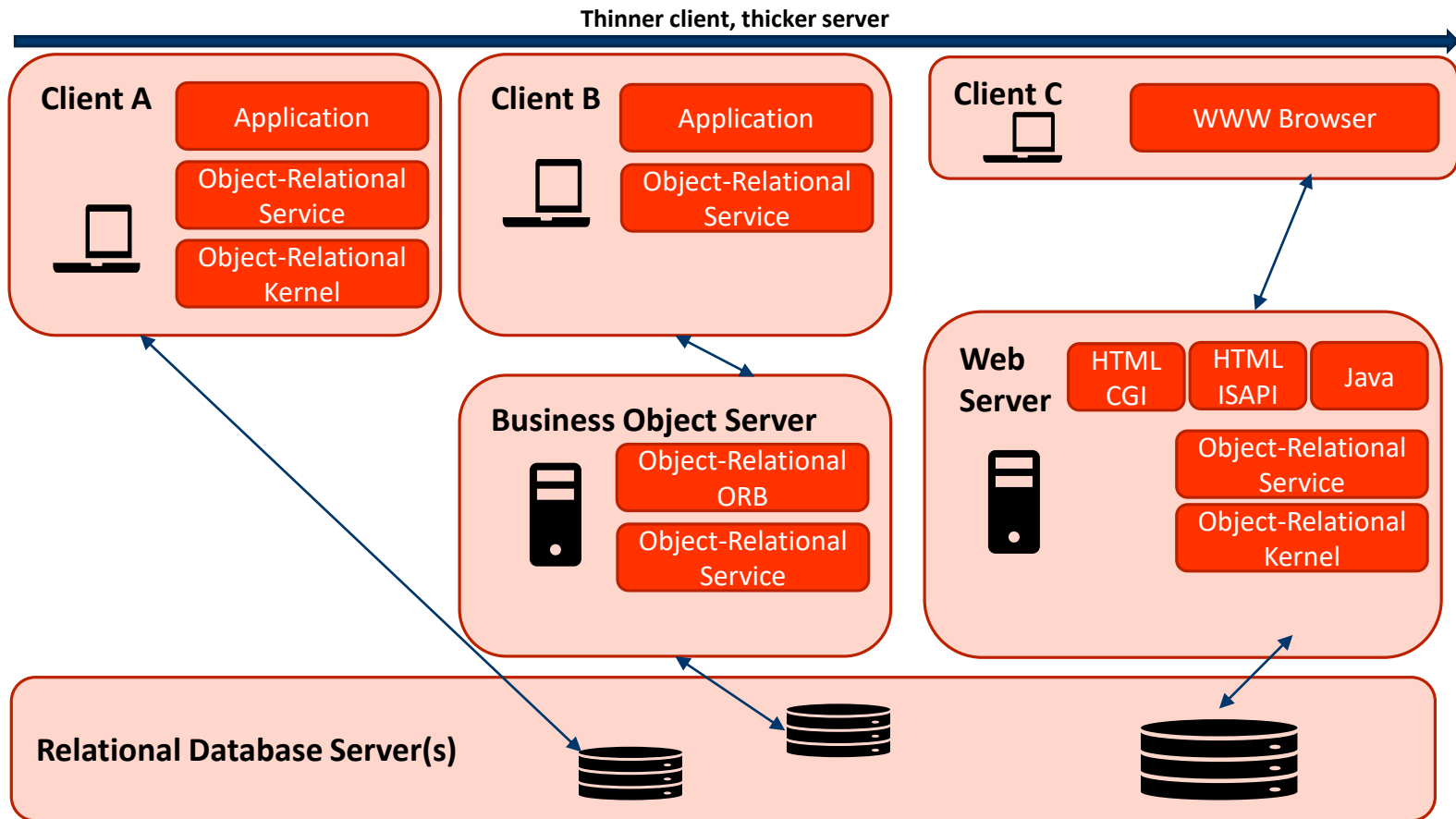
Example: In hospital data processing, a client computer can be running an application program for entering patient information while the server computer is running another program that manages the database in which the information is permanently stored.

Sources: <https://www.britannica.com/technology/client-server-architecture>

Thick-Thin Client

Definition:

A **thin client** is a lightweight computer that has been optimized for establishing a remote connection with a server-based computing environment. The server does most of the work, which can include launching software programs, performing calculations, and storing data.



Source: https://en.wikipedia.org/wiki/Thin_client

3-Tier Architecture

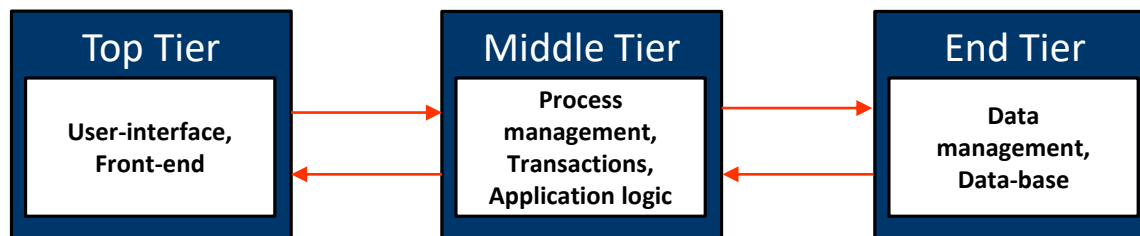
Definition: Multitier_architecture (often referred to as **n-tier architecture**) or **multilayered architecture** is :

A client–server architecture in which presentation, application processing, and data management functions are physically separated.

The most widespread use of **multitier architecture** is the **three-tier architecture**. *N*-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application.

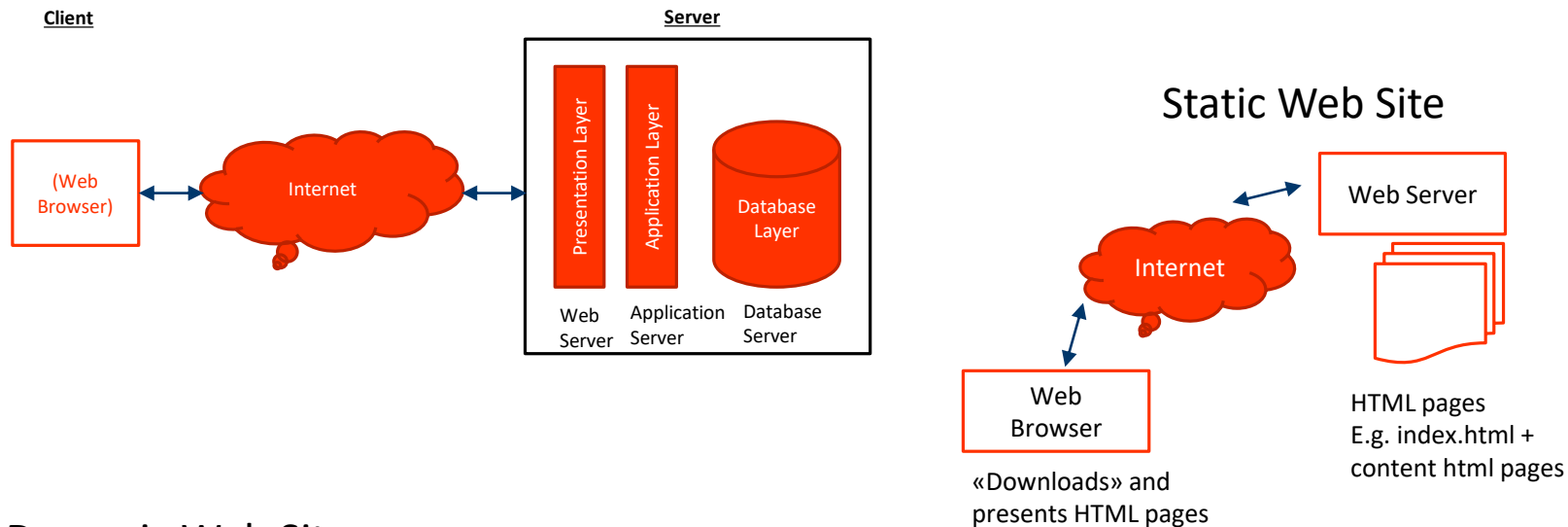
A three-tier architecture is typically composed of a *presentation* tier, a *domain logic* tier, and a *data storage* tier. While the concepts of layer and tier are often used interchangeably, one fairly common point of view is that there is indeed a difference. This view holds that a *layer* is a logical structuring mechanism for the elements that make up the software solution, while a *tier* is a physical structuring mechanism for the system infrastructure. For example, a three-layer solution could easily be deployed on a single tier, such as a personal workstation.

- **Client–server** software architecture pattern
- Developed by **John J. Donovan** in Open Environment Corporation (OEC), a tools company in Cambridge, Massachusetts.
- Indented to allow any of the three tiers to be upgraded or replaced independently in response to changes in requirements or technology.
 - **Presentation tier** (Top tier): it is a layer which users can access directly (such as a web page, or an operating system's GUI).
 - **Application tier** (business logic, logic tier, or middle tier): Controls an application's functionality by performing detailed processing.
 - **Data tier** (end tier): includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. improved scalability and maintainability.

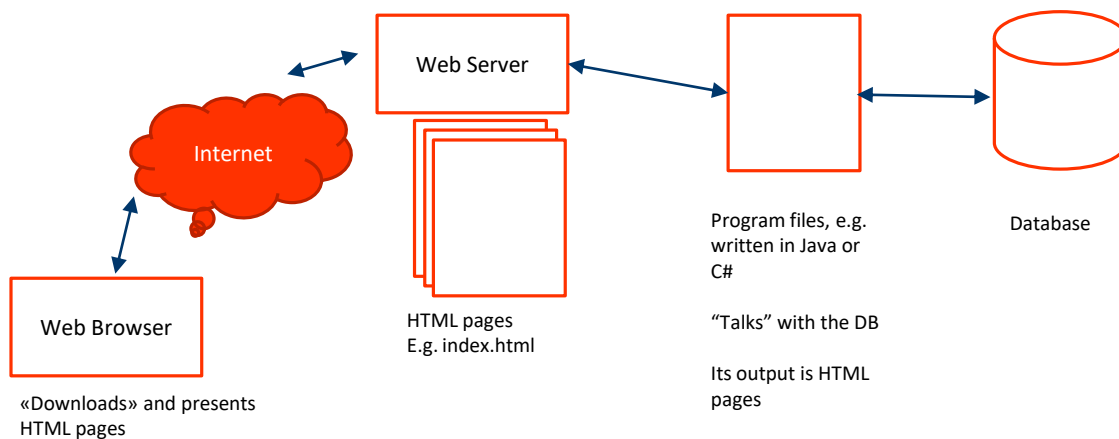


Source: https://en.wikipedia.org/wiki/Multitier_architecture

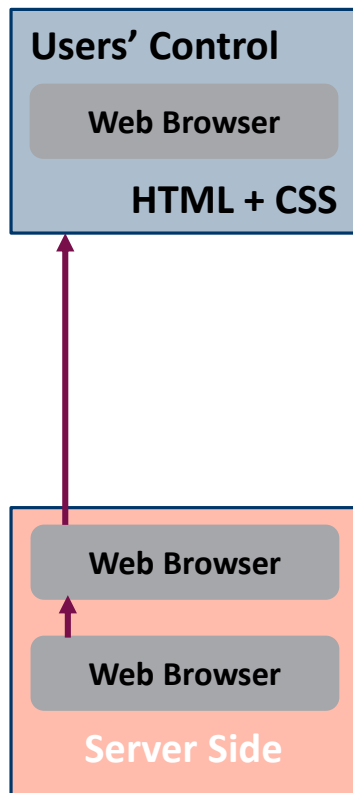
Typical Web Application Architecture



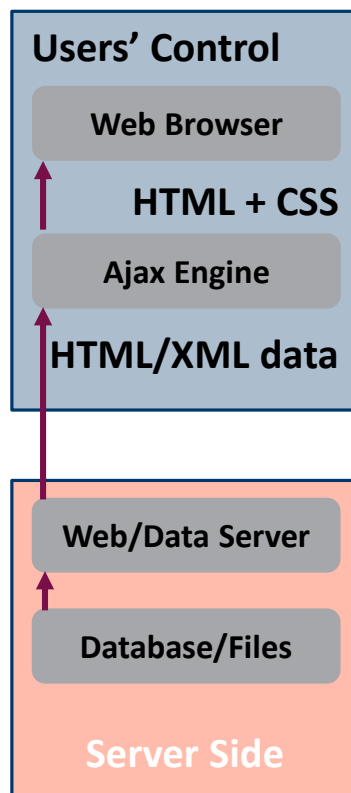
Dynamic Web Site



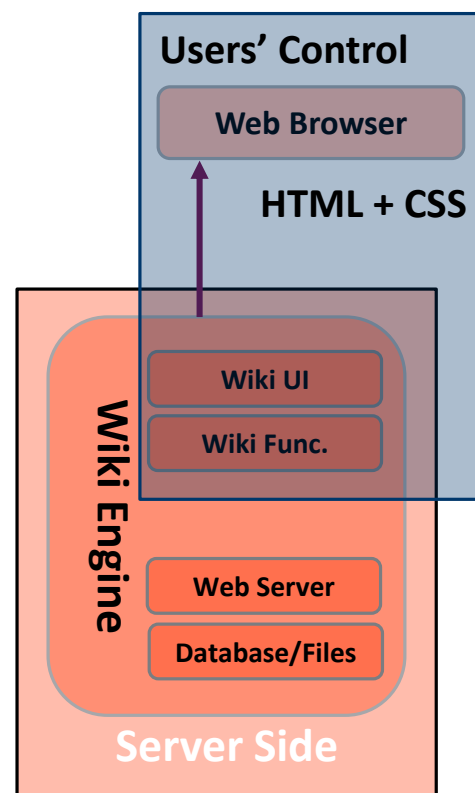
Web Application Model



Conventional Model



AJAX Model



Wiki-based Model

End-to-End Workflow Of A Web Request

DNS Lookup: The client tries to resolve the domain name for the request.

- Client sends DNS Query to local ISP DNS server.
- DNS server responds with the IP address for hostname.com

Connect: Client establishes TCP connection with the IP address of hostname.com

- Client sends SYN packet.
- Web server sends SYN-ACK packet.
- Client answers with ACK packet, concluding the three-way TCP connection establishment.

Send: Client sends the HTTP request to the web server.

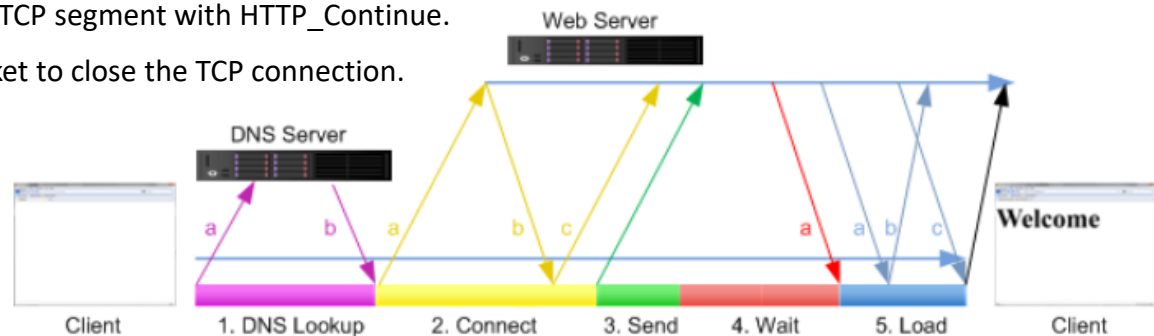
Wait: Client waits for the server to respond to the request.

- Web server processes the request, finds the resource, and sends the response to the Client. Client receives the first byte of the first packet from the web server, which contains the HTTP Response headers and content.

Load: Client loads the content of the response.

- Web server sends second TCP segment with the PSH flag set.
- Client sends ACK. (Client sends ACK every two segments it receives. from the host)
- Web server sends third TCP segment with HTTP_Continue.

Close: Client sends a a FIN packet to close the TCP connection.



Source: <http://www.steves-internet-guide.com/dns-lookups/>

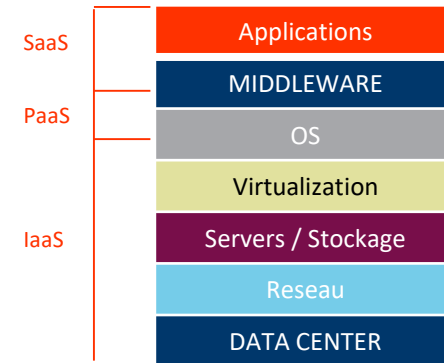
Software-as-a-Service (SaaS) Concept

The term "**Software as a Service**" (**SaaS**) is considered to be part of the nomenclature of **cloud computing** and regards:

- A software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted
- A common delivery model for many business applications, including office software, messaging software, DBMS software, CRM software and many more
- The vast majority of SaaS solutions are based on a multitenant architecture. With this model, a single version of the application, with a single configuration (hardware, network, operating system), is used for all customers ("tenants"). To support scalability, the application is installed on multiple machines (called horizontal scaling).
- In some cases, a second version of the application is set up to offer a select group of customers with access to pre-release versions of the applications (e.g., a beta version) for testing purposes. This is contrasted with traditional software, where multiple physical copies of the software — each potentially of a different version, with a potentially different configuration, and often customized — are installed across various customer sites. In this traditional model, each version of the application is based on a unique code.
- Although an exception rather than the norm, some SaaS solutions do not use multitenancy, or use other mechanisms—such as virtualization—to cost-effectively manage a large number of customers in place of multitenancy. Whether multitenancy is a necessary component for software-as-a-service is a topic of controversy.

There are **two main varieties** of SaaS:

- **Vertical SaaS:** A Software which answers the needs of a specific industry (e.g., software for the healthcare, agriculture, real estate, finance industries)
- **Horizontal SaaS:** The products which focus on a software category (marketing, sales, developer tools, HR) but are industry agnostic.



Source: https://en.wikipedia.org/wiki/Software_as_a_service

Centralized Vs Distributed Application Architectures

Definition:

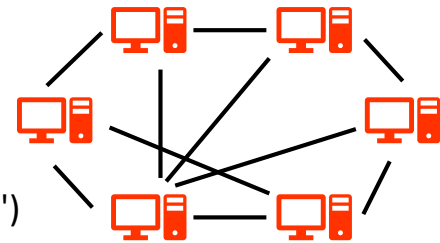
A **distributed system** is a **system** whose components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another. The components interact with one another in order to achieve a common goal.

We need to distinguish between:

- peer-to-peer, and
- microservices

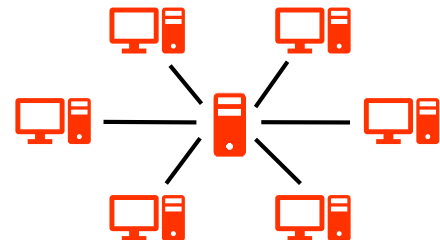
Peer-to-peer

A **peer-to-peer (P2P) network** in which interconnected nodes ("peers") share resources amongst each other without the use of a centralized administrative system



Centralized Network

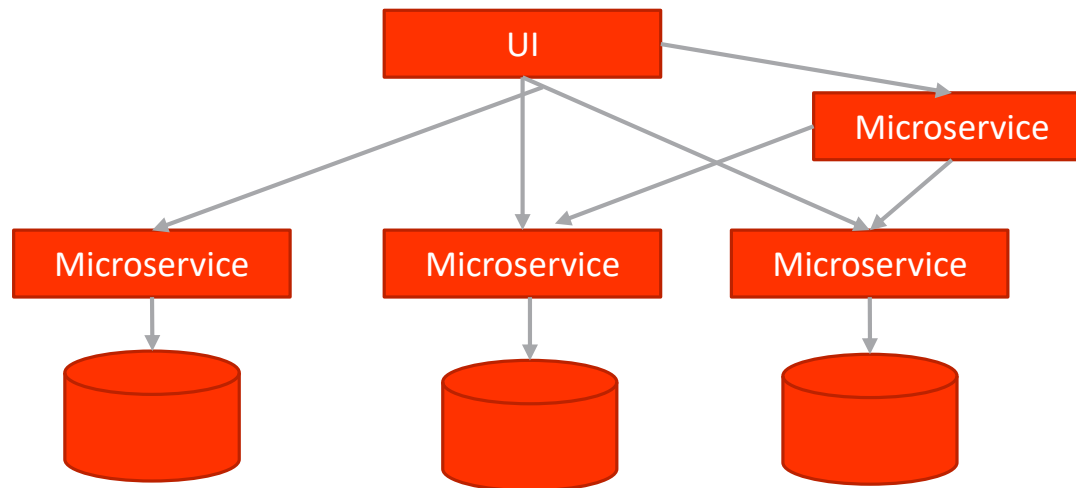
A network based on the client-server model, where individual clients request services and resources from centralized servers



Sources: <https://en.wikipedia.org/wiki/Peer-to-peers>
https://en.wikipedia.org/wiki/Distributed_computing

Microservices

- **Variant** of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.
- In a microservices architecture, services should be **fine - grained** and the protocols should be **lightweight**.
- The **benefit** of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test.
- Parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.
- Allows the architecture of an individual service to emerge through continuous refactoring.
- Microservices-based architectures enable continuous delivery and deployment



Sources: <https://www.redhat.com/en/topics/microservices/what-are-microservices>
<https://en.wikipedia.org/wiki/Microservices>

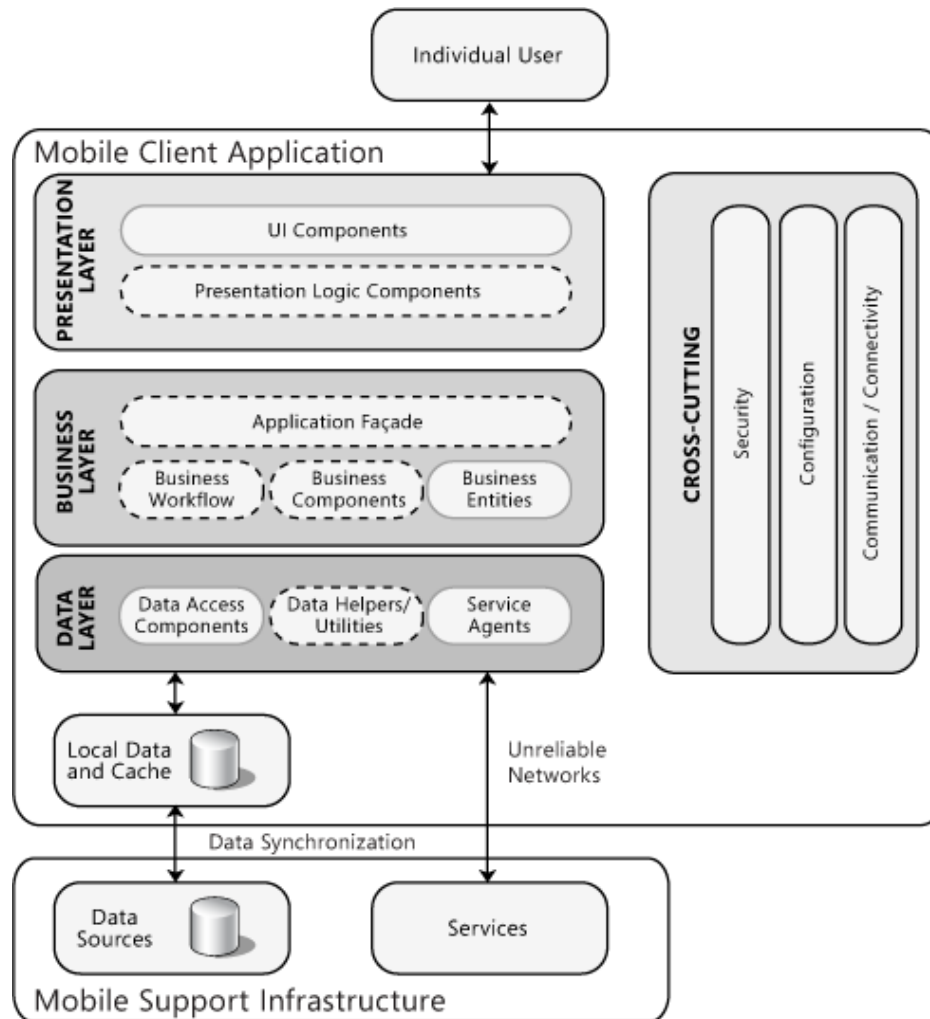
Mobile Application Architectures

- **Scalability** – A Mobile Architecture must be able to be utilized with all recovery requirements on both large and small scale.
- **Secure** – Encryption is important, transmission protocols must support encryption (SSL) via secure transit such as HTTPS
- **Reliable** – Reliability is always important in all technologies and mobile architecture is no different.



Sources: http://www.theshulers.com/whitepapers/mobile_architecture/index.html
https://en.wikipedia.org/wiki/Mobile_architecture

Mobile Apps



Source: https://www.researchgate.net/figure/Layered-Architecture-of-Mobile-Application-3_fig1_328416556



The Architecture of Popular Applications

Examples:

- Amazon is a web-based application
- Angry Birds on your phone is a mobile app
- An ATM's application is client server

Could you name other examples??

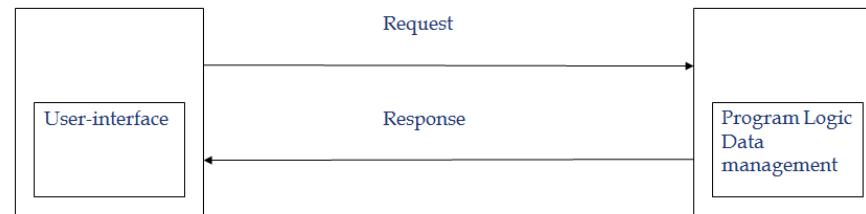
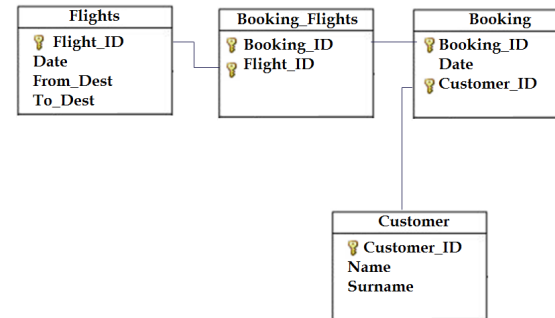


It is time for a Knowledge Check!



Sample Questions

- The capability provided to the consumer to use a provider's applications running on a **cloud infrastructure** is also known as:
 - Platform as a Service (PaaS)
 - Software as a Service (SaaS)
 - Infrastructure as a Service (IaaS)
 - Communication as a Service (CaaS)
- The image on the right shows a:
 - Class Diagram
 - Database Schema
 - Component Diagram
 - User Interface Mockup
- What kind of architecture is shown on the image on the right?
 - Client-Server
 - Thick Client
 - 3-Tier Architecture
 - Web application





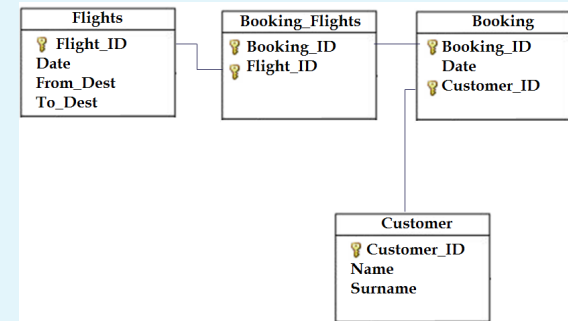
Sample Questions | Answers

1. The capability provided to the consumer to use a provider's applications running on a **cloud infrastructure** is also known as:

- A. Platform as a Service (PaaS)
- B. Software as a Service (SaaS)**
- C. Infrastructure as a Service (IaaS)
- D. Communication as a Service (CaaS)

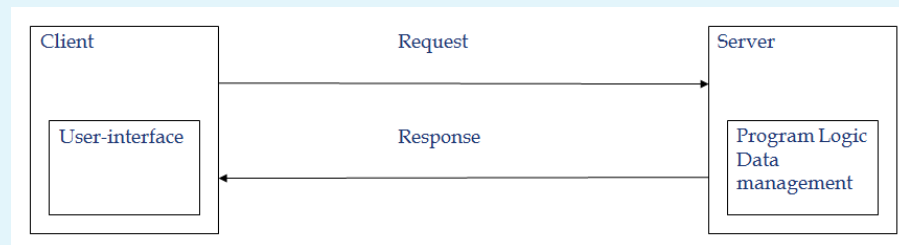
2. The image on the right shows a:

- A. Class Diagram
- B. Database Schema**
- C. Component Diagram
- D. User Interface Mockup



3. What kind of architecture is shown on the image on the right?

- A. Client-Server**
- B. Thick Client
- C. 3-Tier Architecture
- D. Web application



Coding Bootcamp

Lesson 6

Software Development Methodologies

Objectives:

- Application Development Process
- Software Development Methodologies
- Agile Software Development
- How Code Reviews Work
- How DevOps and Scrum Work

Syllabus Items:

- 1.3 Software Development Methodologies



Syllabus

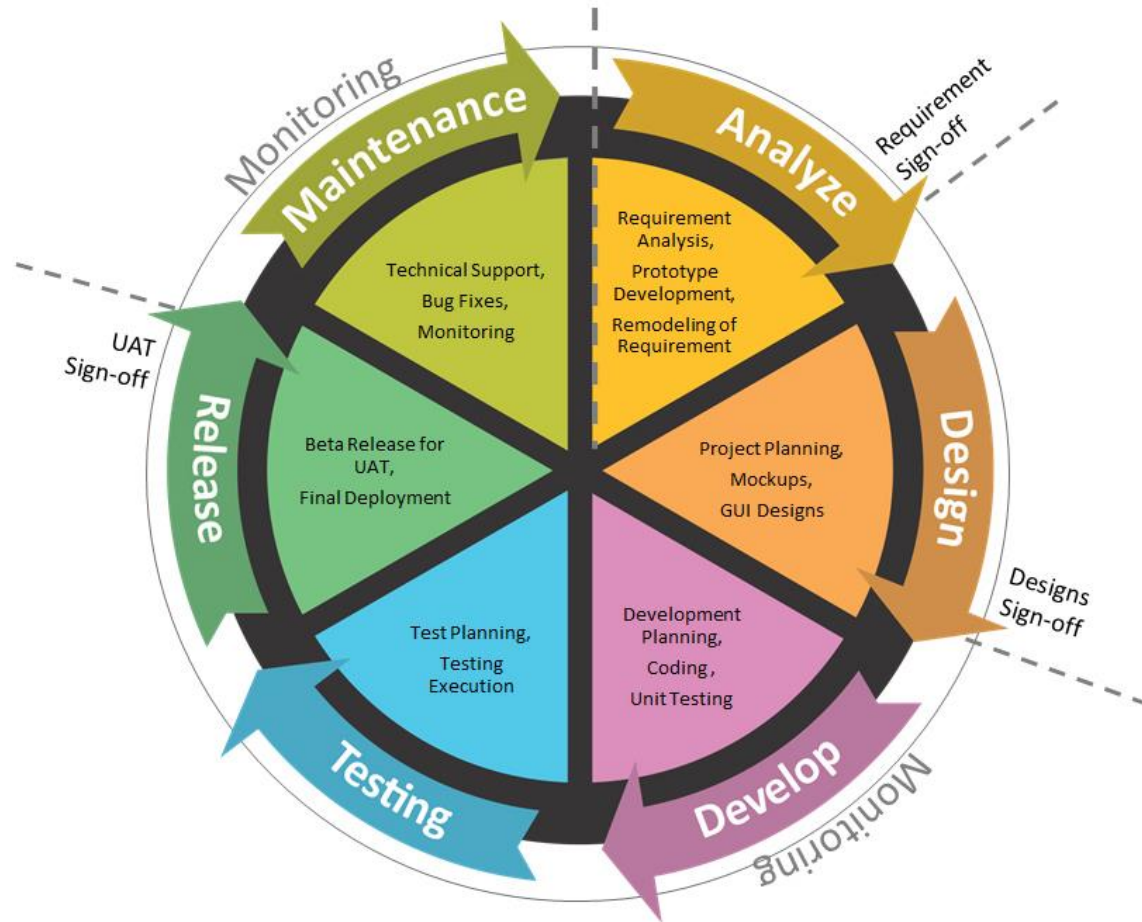
Category	Topic	Task
FSD_1 Software Design and Development	1.3 Software Development Methodologies	1.3.1 Understand the application development process
		1.3.2 Describe the main software development methodologies, like: waterfall, prototyping, rapid application, agile code and fix. Distinguish between these methodologies and when they are applied.
		1.3.3 List the main software development related activities as per software methodology: Requirements, Design, Construction, Testing, Debugging, Deployment, Maintenance
		1.3.4 Understand how code reviews work
		1.3.5 Understand how DevOps works
		1.3.6 Understand how Scrum works



Contents | Learning Objectives

- ✓ Understand the Application Development process along with the methodologies that can be used
- ✓ Know and describe how the most commonly used software development methodologies work and when they should be applied
- ✓ Know which are the main software development activities and what is included in each activity
- ✓ Understand what is a code review and how it is used
- ✓ Understand what is DevOps and Scrum and how they can be used

Application Development Process



Source: <https://simplified-it-outsourcing.com/offshore-software-development-methodologies>



Software Development Methodologies, Paradigms and Models

- Waterfall
- Prototyping
- Rapid application
- Software engineering
- Waterfall
- Prototyping
- Iterative and incremental development (IID)
 - Incremental
 - Spiral
- V-Model, Dual Vee Model
- Agile (2001)
- Lean (2003)
- DevOps (2008)
- Distinguish between these methodologies and when they are applied

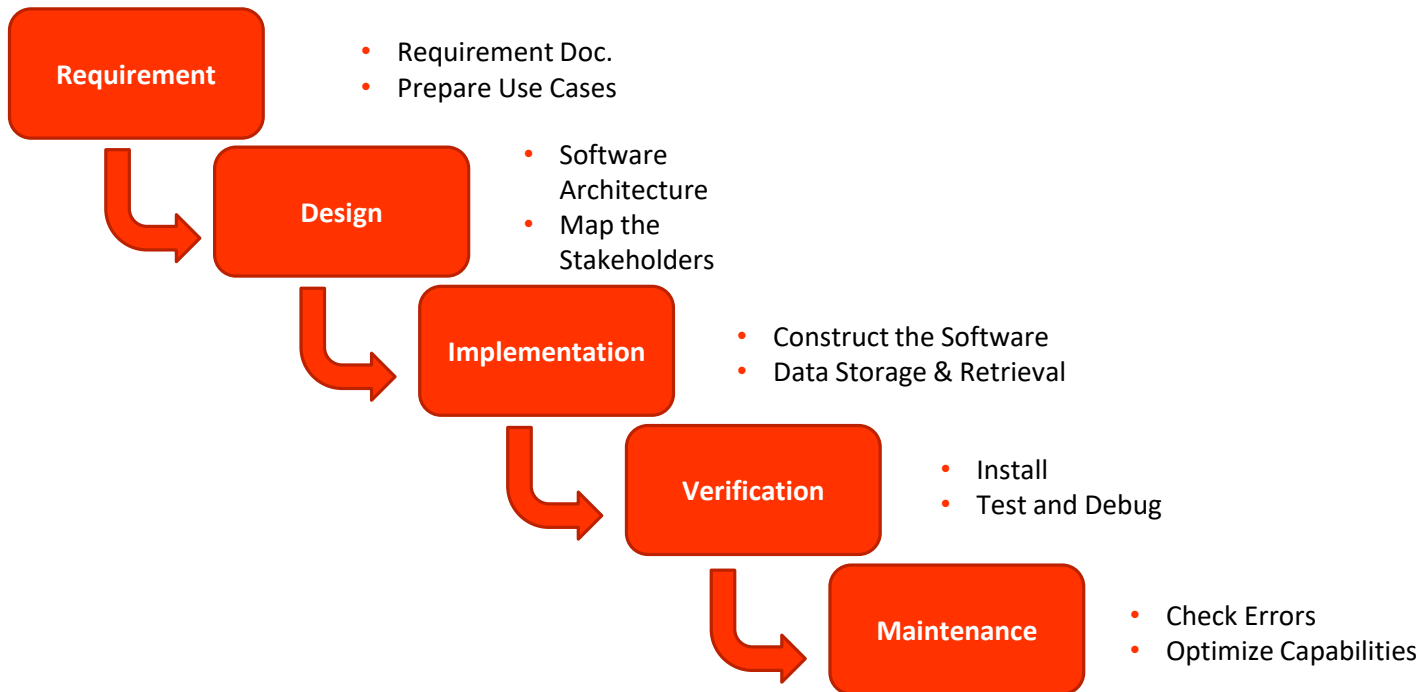
Waterfall

Definition:

The **waterfall model** is a breakdown of project activities into linear **sequential** phases, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks.

- In software development, it is considered a less iterative and flexible approach, as progress flows in largely one direction, "downwards" like a waterfall, through the phases of conception, initiation, analysis, design, construction, testing, deployment and maintenance.

Main phases: Requirements, Design, Construction, Testing, Debugging, Deployment, Maintenance



Source: https://en.wikipedia.org/wiki/Waterfall_model



Waterfall Model Phases

Requirements Analysis and Definition

- System's services, constraints and goals = System specification

System and Software Design

- Partitions the requirements to either Software or Hardware systems.
- System architecture

Implementation and Unit Testing

- The Software design is realised as a set of programs or program units

Integration and System Testing

Operation and Maintenance

- **The result of each phase is one or more documents which are approved ("signed off")**

Waterfall Model Problems

- Inflexible partitioning of the project into distinct stages
- This makes it difficult to respond to changing customer requirements
- Therefore, this model is only appropriate when the requirements are well-understood
- **The drawback of the waterfall model is the difficulty of accommodating change after the process is underway**

Source: <https://iansommerville.com/software-engineering-book/>



Prototyping

Definition:

Software **prototyping** is the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed

- Comparable to prototyping as in mechanical engineering or manufacturing.
- A prototype typically simulates only a few aspects of, and may be completely different from, the final product.

Prototyping Process (iterative process for steps #3 and #4):

1. **Identify basic requirements:** gather input and output information desired.
2. **Develop initial prototype:** includes only user interfaces.
3. **Review:** customers (plus end-users) review the prototype and provide feedback
4. **Revise and enhance the prototype:** specifications and prototype can be improved, while negotiation about what is within the scope of the project may be necessary.

Benefits of Prototyping:

- ✓ **User involvement:** Valuable use feedback is gathered early in the project
- ✓ **Reduced time and costs:** The client and the contractor can compare if the software made matches the software specification. The software engineer gains insight into the accuracy of initial project estimates and whether the deadlines and milestones proposed can be successfully met

Disadvantages of Prototyping:

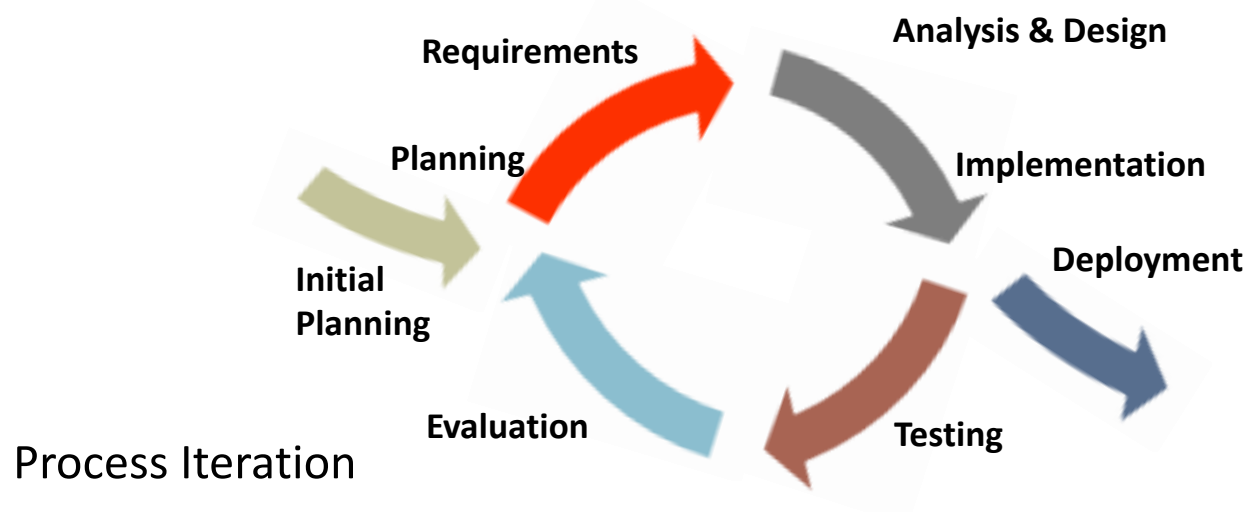
- **Insufficient analysis:** The focus on a limited prototype can distract developers from properly analyzing the complete project
- **User confusion of prototype and finished system:** Users can begin to think that a prototype, intended to be thrown away, is actually a final system that merely needs to be finished or polished.
- **Developer misunderstanding of user objectives:** Developers may assume that users share their objectives, without understanding wider commercial issues.

Source: https://en.wikipedia.org/wiki/Software_prototyping

Iterative And Incremental Development (IID)

Definition:

Iterative and incremental software development is a method of software development that is modeled around a gradual increase in feature additions and a cyclical release and upgrade pattern.

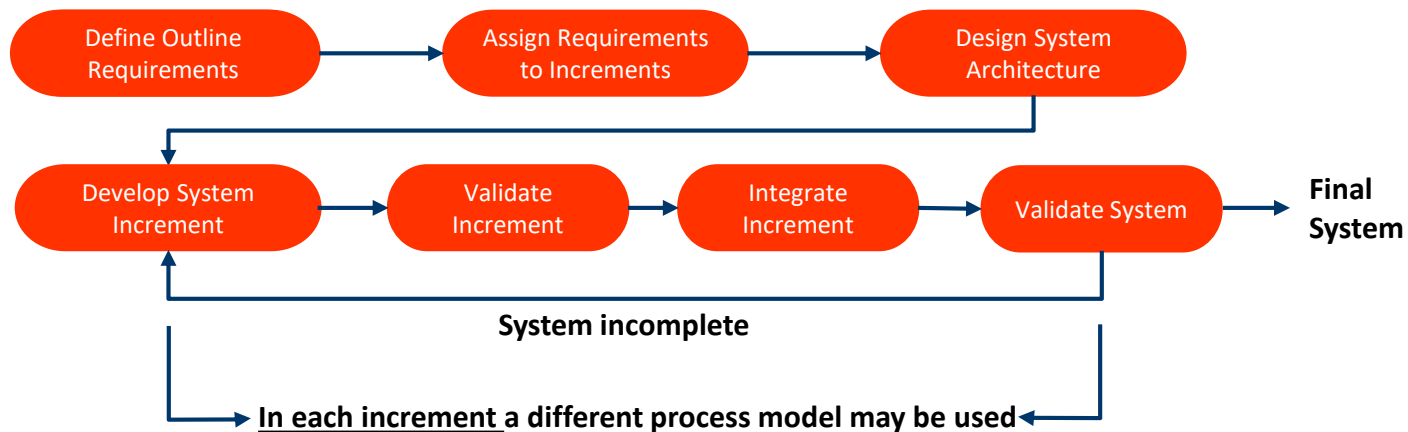


- System requirements **ALWAYS** evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems
- **Iteration** can be applied to any of the generic process models
- Two (related) approaches to IID
 - **Incremental development**
 - **Spiral development**

Sources: <https://www.techopedia.com/definition/25895/iterative-and-incremental-development>

Incremental Development (Mills, 1980)

- 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 are frozen though requirements for later increments can continue to evolve



Source: <https://iansommerville.com/software-engineering-book/>



Incremental Development | Advantages | Disadvantages

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

Disadvantages:

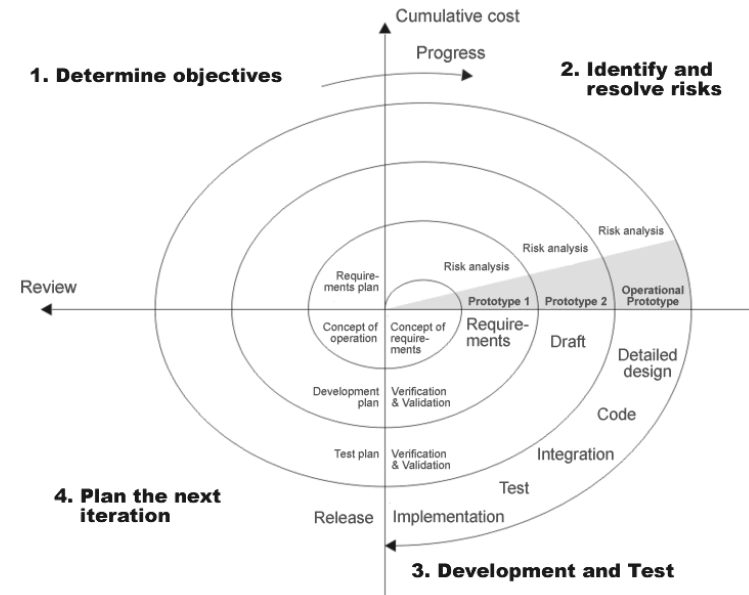
- Increments should be relatively small (< 20.000 LOC*)
- Each increment should deliver some functionality
- Difficult to map customer's requirements onto increments of the right size

* LOC → *Lines of Code*

Source: <https://iansommerville.com/software-engineering-book/>

Spiral Development (Boehm, 1988)

- Process is represented as a **spiral** rather than as a **sequence** of activities with backtracking
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required
- Risks are explicitly assessed and resolved throughout the process



Spiral Model Sectors

Objective setting

- Specific objectives for the phase are identified

Risk assessment and reduction (distinguishes it from other models)

- Risks are assessed and activities put in place to reduce the key risks

Development and validation

- A development model for the system is chosen which can be any of the generic models

Planning

- The project is reviewed, and the next phase of the spiral is planned

Source: <https://iansommerville.com/software-engineering-book/>



Rapid Application Development (RAD)

Definition:

Rapid-application development (RAD) is both:

A general term used to refer to alternatives to the conventional waterfall model of software development; and the name for James Martin's approach to rapid development.

- RAD approaches to Software Development (SD) put less emphasis on planning and more emphasis on process.
- RAD approaches emphasize adaptability and the necessity of adjusting requirements in response to knowledge gained as the project progresses.
- Prototypes are often used in addition to or sometimes even in place of design specifications.
- RAD is especially well suited for (although not limited to) developing software that is driven by user interface requirements.
- Other approaches to rapid development include Agile methods and the spiral model

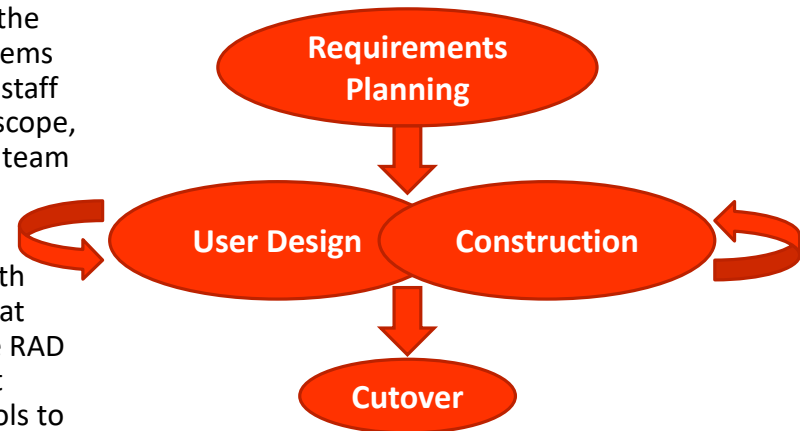
In contrast to the waterfall model, which calls for rigorously defined specification to be established prior to entering the development phase,

Sources: <https://iansommerville.com/software-engineering-book/>
https://en.wikipedia.org/wiki/Rapid_application_development

James Martin's Approach To RAD

The **James Martin** approach to RAD divides the process into **four** distinct phases:

1. **Requirements planning phase** – combines elements of the system planning and systems analysis phases of the Systems Development Life Cycle (SDLC). Users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements. It ends when the team agrees on the key issues and obtains management authorization to continue.
2. **User design phase** – during this phase, users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. *User Design* is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.
3. **Construction phase** – focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.
4. **Cutover phase** – resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner.



Sources: <https://iansommerville.com/software-engineering-book/>
https://en.wikipedia.org/wiki/Rapid_application_development



RAD – Advantages and Disadvantages (Pros and Cons)

The **advantages (pros)** of RAD include:

- **Better quality:** By having users interact with evolving prototypes the business functionality from a RAD project can often be much higher than that achieved via a waterfall model. The software can be more usable and has a better chance to focus on business problems that are critical to end users rather than technical problems of interest to developers.
- **Risk control:** Although much of the literature on RAD focuses on speed and user involvement a critical feature of RAD done correctly is risk mitigation. It's worth remembering that Boehm initially characterized the spiral model as a risk based approach. A RAD approach can focus in early on the key risk factors and adjust to them based on empirical evidence collected in the early part of the process. E.g., the complexity of prototyping some of the most complex parts of the system.
- **More projects completed on time and within budget:** By focusing on the development of incremental units the chances for catastrophic failures that have dogged large waterfall projects is reduced. In the Waterfall model it was common to come to a realization after six months or more of analysis and development that required a radical rethinking of the entire system. With RAD this kind of information can be discovered and acted upon earlier in the process.

The **disadvantages (cons)** of RAD include:

- **The risk of a new approach:** For most IT shops RAD was a new approach that required experienced professionals to rethink the way they worked. Humans are virtually always averse to change and any project undertaken with new tools or methods will be more likely to fail the first time simply due to the requirement for the team to learn.
- **Requires time of scarce resources:** One thing virtually all approaches to RAD have in common is that there is much more interaction throughout the entire life-cycle between users and developers. In the waterfall model, users would define requirements and then mostly go away as developers created the system. In RAD users are involved from the beginning and through virtually the entire project. This requires that the business is willing to invest the time of application domain experts. The paradox is that the better the expert, the more they are familiar with their domain, the more they are required to actually run the business and it may be difficult to convince their supervisors to invest their time. Without such commitments RAD projects will not succeed.
- **Less control:** One of the advantages of RAD is that it provides a flexible adaptable process. The ideal is to be able to adapt quickly to both problems and opportunities. There is an inevitable trade-off between flexibility and control, more of one means less of the other. If a project (e.g. life-critical software) values control more than agility RAD is not appropriate.
- **Poor design:** The focus on prototypes can be taken too far in some cases resulting in a "hack and test" methodology where developers are constantly making minor changes to individual components and ignoring system architecture issues that could result in a better overall design. This can especially be an issue for methodologies such as Martin's that focus so heavily on the user interface of the system.
- **Lack of scalability:** RAD typically focuses on small to medium-sized project teams. The other issues cited above (less design and control) present special challenges when using a RAD approach for very large-scale systems.

Source: https://en.wikipedia.org/wiki/Rapid_application_development



Agile Software Development

Definition:

Agile software development comprises various approaches to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customer(s)/end user(s).

- It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change.
- The term *agile* (sometimes written *Agile*) was popularized, in this context, by the *Manifesto for Agile Software Development*.
- The values and principles espoused in this manifesto were derived from and underpin a broad range of software development frameworks, including Scrum and Kanban.
- Agile software development describes a set of principles for software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams
- First coined in 2001, in the *Manifesto for Agile Software Development*

Sources: https://en.wikipedia.org/wiki/Agile_software_development
<https://agilemanifesto.org/>



Manifesto for Agile Software Development

- **Individuals and Interactions** more than processes and tools: Self-organization and motivation are important, as are interactions like co-location and pair programming. It is better to have a good team of developers who communicate and collaborate well, rather than a team of experts each operating in isolation. Communication is a fundamental concept.
- **Working Software** more than comprehensive documentation: Working software is more useful and welcome than just presenting documents to clients in meetings. It is better to comment inline with the code and to keep external documentation light, rather than heavy documents that take a lot of effort and quickly become outdated.
- **Customer Collaboration** more than contract negotiation: Requirements cannot be fully collected at the beginning of the software development cycle, so it is better to directly involve the paying customer and their end-users, or a proxy for them so that detailed requirements can be progressively elaborated and adapted based on feedback.
- **Responding to Change** more than following a plan: Agile software development methods are focused on quick responses to change and continuous development. Some of the authors formed the Agile Alliance, a non-profit organization that promotes software development according to the manifesto's values and principles. Introducing the manifesto on behalf of the Agile Alliance, Jim Highsmith said,
- The Agile movement is not anti-methodology, in fact many of us want to restore credibility to the word methodology. We want to restore a balance. We embrace modeling, but not in order to file some diagram in a dusty corporate repository. We embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes. We plan, but recognize the limits of planning in a turbulent environment. Those who would brand proponents of XP or SCRUM or any of the other Agile Methodologies as "hackers" are ignorant of both the methodologies and the original definition of the term hacker. — *Jim Highsmith, History: The Agile Manifesto*

Sources: https://en.wikipedia.org/wiki/Agile_software_development
<https://agilemanifesto.org/>



Agile Development | 12 Principles

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business-people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Sources: https://en.wikipedia.org/wiki/Agile_software_development
<https://agilemanifesto.org/principles.html>



Agile Development | Pitfalls

- Lack of overall product design
- Adding stories to an iteration in progress
- Lack of sponsor support
- Insufficient training
- Product owner role is not properly filled
- Teams are not focused
- Excessive preparation/planning
- Problem-solving in the daily standup
- Fixed time, resources, scope, and quality
- Attempting to take on too much in an iteration
- Allowing technical debt to build up

Sources: https://en.wikipedia.org/wiki/Agile_software_development
<https://agilemanifesto.org/principles.html>



Lean

Definition:

Lean software development (LSD) is a translation of lean manufacturing and lean IT principles and practices to the software development domain

- Adapted from the Toyota Production System, it emerged from within the Agile community

Lean principles

- **Eliminate waste:** Lean philosophy regards everything not adding value to the customer as waste (muda). Such waste for software development may include: building the wrong feature or product, unnecessarily complex solutions, rework, ineffective communication etc.
- **Amplify learning:** For example, the learning process can be sped up by usage of short iteration cycles – each one coupled with refactoring and integration testing.
- **Decide as late as possible:** move the building of options earlier for customers, thus delaying certain crucial decisions until customers have realized their needs better
- **Deliver as fast as possible:** The sooner the end product is delivered without major defects, the sooner feedback can be received, and incorporated into the next iteration. The shorter the iterations, the better the learning and communication within the team.
- **Empower the team:** build projects around motivated individuals and trust them to get the job done; respecting people and acknowledging their work is one way to empower the team.
- **Build integrity in:** Conceptual integrity means that the system's separate components work well together as a whole with balance between flexibility, maintainability, efficiency, and responsiveness.
- **Optimize the whole:** "Think big, act small, fail fast; learn rapidly"; Only when all of the lean principles are implemented together, combined with strong "common sense" with respect to the working environment, is there a basis for success in software development.

Source: https://en.wikipedia.org/wiki/Lean_software_development

How Code Reviews Work

Definition:

Code review is “a process where two or more developers visually inspect a set of program code, typically, several times”

Code review objectives include: Best Practice, Error Detection, Vulnerability Exposure and Malware Discovery

Two main categories of code review:

1. **Formal code review:** such as a Fagan inspection, involves a careful and detailed process with multiple participants and multiple phases. Formal code reviews are the traditional method of review, in which software developers attend a series of meetings and review code line by line, usually using printed copies of the material. Formal inspections are extremely thorough and have been proven effective at finding defects in the code under review.
2. **Lightweight code review:** typically requires less overhead than formal code inspections, though it can be equally effective when done properly. Lightweight reviews are often conducted as part of the normal development process:
 - Formal code review, Lightweight code review Over-the-shoulder – one developer looks over the author's shoulder as the latter walks through the code.
 - Email pass-around – source code management system emails code to reviewers automatically after check-in is made.
 - Pair programming – two authors develop code together at the same workstation, as is common in Extreme Programming.
 - Tool-assisted code review – authors and reviewers use software tools, informal ones such as pastebins and IRC, or specialized tools designed for peer code review.

Some of these are also known as walkthrough (informal) or "critique" (fast and informal) code review types.

Many teams that eschew traditional, formal code review use one of the above forms of lightweight review as part of their normal development process. A code review case study published in the book *Best Kept Secrets of Peer Code Review* found that lightweight reviews uncovered as many bugs as formal reviews, but were faster and more cost-effective.

- **Code Review Saves Time and Cost!!!**

Extra Resources on code review:

- <http://www.methodsandtools.com/archive/archive.php?id=66>
- <http://www.ganssle.com/inspections.pdf>

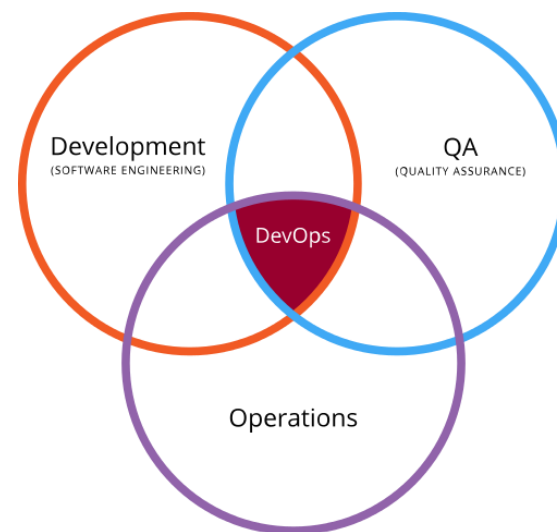
Sources: https://en.wikipedia.org/wiki/Code_review
https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Quality/Code_Review

How DevOps Works

Definition:

DevOps (a clipped compound of Development and Operations) is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes

- DevOps is a set of practices that combines software development (Dev) and information-technology operations (Ops) which aims to shorten the systems development life cycle and provide continuous delivery with high software quality
- As DevOps is intended to be a cross-functional mode of working, those that practice the methodology use different sets of tools—referred to as "toolchains"—rather than a single one
- Toolchains are used for:
 - Coding
 - Building (continuous integration tools)
 - Testing (continuous testing tools)
 - Packaging (artifact repository)
 - Releasing (change management, release automation)
 - Configuring
 - Monitoring



Sources: PeopleCert DevOps Fundamentals courseware
<https://en.wikipedia.org/wiki/DevOps>



How Scrum Works

Definition:

Scrum is an iterative and incremental agile software development framework for managing product development

Scrum:

- defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal"
- challenges assumptions of the "traditional, sequential approach" to product development
- enables teams to self-organize by encouraging physical co-location or close online collaboration of all team members, as well as daily face-to-face communication among all team members and disciplines involved

More on Scrum on a dedicated lesson in the upcoming classes of your Coding Bootcamp!

Source: <https://commons.wikimedia.org/w/index.php?curid=20202905>



It is time for a Knowledge Check!



Sample Questions

1. _____ defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal"
 - A. Scrum
 - B. DevOps
 - C. Agile
 - D. Project Management

2. RAD stands for:
 - A. Rapid Agile Development
 - B. Random Application Design
 - C. Rapid Application Development
 - D. Random Agile Design

3. A splitting of software development work into distinct phases (or stages) containing activities with the intent of better planning and management describes:
 - A. Requirements capturing
 - B. Project management
 - C. Software as a Service
 - D. Application development



Sample Questions | Answers

1. _____ defines "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal"
 - A. **Scrum**
 - B. DevOps
 - C. Agile
 - D. Project Management

2. RAD stands for:
 - A. Rapid Agile Development
 - B. Random Application Design
 - C. **Rapid Application Development**
 - D. Random Agile Design

3. A splitting of software development work into distinct phases (or stages) containing activities with the intent of better planning and management describes:
 - A. Requirements capturing
 - B. Project management
 - C. Software as a Service
 - D. **Application development**



Any Questions?



References | Further Reading

- <http://fiftyexamples.readthedocs.io/en/latest/algorithms.html>
- https://en.wikipedia.org/wiki/Software_architecture
- <https://www.britannica.com/technology/client-server-architecture>
- https://en.wikipedia.org/wiki/Thin_client
- https://en.wikipedia.org/wiki/Multitier_architecturemultitier_architecture
- https://en.wikipedia.org/wiki/Software_as_a_service
- <https://en.wikipedia.org/wiki/Peer-to-peer>
- <https://en.wikipedia.org/wiki/Microservices>
- http://www.theshulers.com/whitepapers/mobile_architecture/index.html
- https://en.wikipedia.org/wiki/Mobile_architecture
- <https://simplified-it-outsourcing.com/offshore-software-development-methodologies>
- https://en.wikipedia.org/wiki/Agile_software_development
- https://en.wikipedia.org/wiki/Code_review
- <https://commons.wikimedia.org/w/index.php?curid=20202905>
- https://computersciencewiki.org/index.php/Data_representation
- <https://en.wikipedia.org/wiki/DevOps>
- <http://www.ganssle.com/inspections.pdf>
- https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Quality/Code_Review
- <http://www.methodsandtools.com/archive/archive.php?id=66>
- PeopleCert DevOps Fundamentals courseware
- <http://www.steves-internet-guide.com/dns-lookups/>
- <https://home.adelphi.edu/~siegfried/cs170/170l1.pdf>
- https://en.wikipedia.org/wiki/Boolean_algebra
- <https://library.alliant.edu/screens/boolean.pdf>
- https://en.wikipedia.org/wiki/Binary_number
- <https://en.wikipedia.org/wiki/Decimal>
- https://en.wikipedia.org/wiki/Computer_memory
- https://en.wikipedia.org/wiki/Application_software
- <https://www.jinfont.com/resources/bi-defined/3-tier-architecture-complete-overview/>
- <https://en.wikipedia.org/wiki/Computation>
- <https://whatis.techtarget.com/definition/algorithm>
- https://en.wikipedia.org/wiki/Computer_program
- <https://www.computerhope.com/jargon/s/software.htm>
- https://en.wikipedia.org/wiki/Distributed_computing
- <https://www.redhat.com/en/topics/microservices/what-are-microservices>
- https://en.wikipedia.org/wiki/Waterfall_model
- https://en.wikipedia.org/wiki/Rapid_application_development
- https://en.wikipedia.org/wiki/Agile_software_development
- <https://agilemanifesto.org/principles.html>
- https://en.wikipedia.org/wiki/Lean_software_development
- <https://www.techopedia.com/definition/25895/iterative-and-incremental-development>
- https://en.wikipedia.org/wiki/Code_review

Thank You!

For latest news and updates follow us.



[linkedin.com/company/peoplecert-group/](https://www.linkedin.com/company/peoplecert-group/)

twitter.com/peoplecert

[youtube.com/channel/UCLBidKZS9Xk08f_PM5Edtfg](https://www.youtube.com/channel/UCLBidKZS9Xk08f_PM5Edtfg)

facebook.com/peoplecert.org