PeopleCert Data science

PeopleCert Data Science

Data Analyst

Three-Day Course

Study Guide

Copyright Details

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

PeopleCert

All talents, certified.

PeopleCert: A Global Leader in Certification

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

AXELOS GLOBAL BEST PRACTICE · PeopleCert DevOps. · PeopleCert Scrum. · IASSC · ISACA · Language Cert · LEAN IT ASSOCIATION · Quality Software Development · ECDL

PeopleCert
All talents, certified.

How to Use This Document

This document is your **PeopleCert Data Science: Data Analyst Study Guide** to help you prepare for the **PeopleCert Data Science: Data Analyst examination (Practitioner level)**.

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 Data Science: Data Analyst exam.

Your exams will be closed book. You will be given 60 minutes to complete it. It contains 40 multiple choice questions and to pass the exam you must achieve a grade of 70% or higher, or a minimum of 28/40 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.

Syllabus 1/2

Category	Topic	Skill Set
1.0 Introduction to Data Science	1.1 Overview & Definitions	1.1.1 General Terms and Definitions
	1.2 Key Concepts	1.2.1 Data Analytics
2.0 Programming Skills (with R/Python)	2.1 Introduction to Programming	2.1.1 Key Concepts
		2.1.2 Developer Tools
	2.2 Basic Programming Skills	2.2.1 Programming Basics
		2.2.2 Algorithm Basics
3.0 Data Management	3.1 Relational Databases (RDBMS)	3.1.1 Key Terms/Definitions
		3.1.2 Database Design
		3.1.3 SQL
	3.2 New Data Management Methods	3.2.1 NoSQL
		3.2.2 Data Lakes
	3.3 Business Intelligence	3.3.1 Key Concepts and Basic Use of PowerBI
		3.3.2 Extract, Transform, Load (ETL)
4.0 Probability & Statistics	4.1 Introduction to Statistics	4.1.2 Descriptive Statistics
	4.2 Advanced Statistical Topics	4.2.1 Time Series

Syllabus 2/2

Category	Topic	Skill Set
5.0 Machine Learning (ML) and Artificial Intelligence (AI)	5.1 Machine Learning (ML)	5.1.1 Introduction to ML
		5.1.2 Algorithms in Machine Learning
	5.2 Artificial Intelligence (AI)	5.2.1 General Concepts of AI
6.0 Visualization	6.1 Introduction to Visualization	6.1.2 Visualization Basics
7.0 Business Skills	7.1 Data Governance	7.1.1 The Need for Data Governance
		7.1.2 Data Governance Strategy
	7.2 Ethics, Data Privacy and Protection	7.2.1 Data Privacy & GDPR
		7.2.2 Anonymize Data

Contents | Learning Objectives

This course aims at obtaining the **PeopleCert Data Science: Data Analyst certification**, and covers the following contents, where students/candidate should be able to demonstrate their knowledge and understanding:

- ✓ The basics of Data Science
- ✓ Basic Programming Skills (with R/Python)
- ✓ Data Management and Relational Databases
- ✓ Business intelligence and PowerBI basics
- ✓ Probability and Statistics
- ✓ The basics of Machine Learning
- ✓ The basics of Artificial Intelligence (AI)
- ✓ Visualization
- ✓ Data Governance, Ethics, Data Privacy and Protection

PeopleCert Data Science
Data Analyst

Objectives:

- The basics of Data Science
- Basic Programming Skills (with R/Python)
- Data Management & Relational Databases
- Business intelligence & PowerBI basics
- Time Series Basics with Power BI
- Probability & Statistics
- Data Governance, Ethics, Data Privacy and Protection
- The basics of Machine Learning
- The basics of Artificial Intelligence

Syllabus Items:

- 1.0 Introduction to Data Science
- 2.0 Programming Skills (with R/Python)
- 3.0 Data Management
- 4.0 Probability & Statistics
- 5.0 Machine Learning and Artificial Intelligence
- 6.0 Visualization
- 7.0 Business Skills

PeopleCert Data Science

Introduction to Programming

Introduction to Programming

- Why do you need programming skils for DS/DA/AI/ML?
  - All of them require important computer power.
  - In order to communicate with a computers, a language is needed.
  - Depending on the goal, you might chose one programming langauge or other.
  - Two languages predominate:
    - **Python**

      General-purpose programming language having multiple data science libraries along with rapid prototyping.

    - **R**

      Language for statistical analysis and visualization.

Source: https://www.analyticsvidhya.com/blog/2020/11/14-must-have-skills-to-become-a-data-scientist-with-resources/

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

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

Algorithm

Definition:

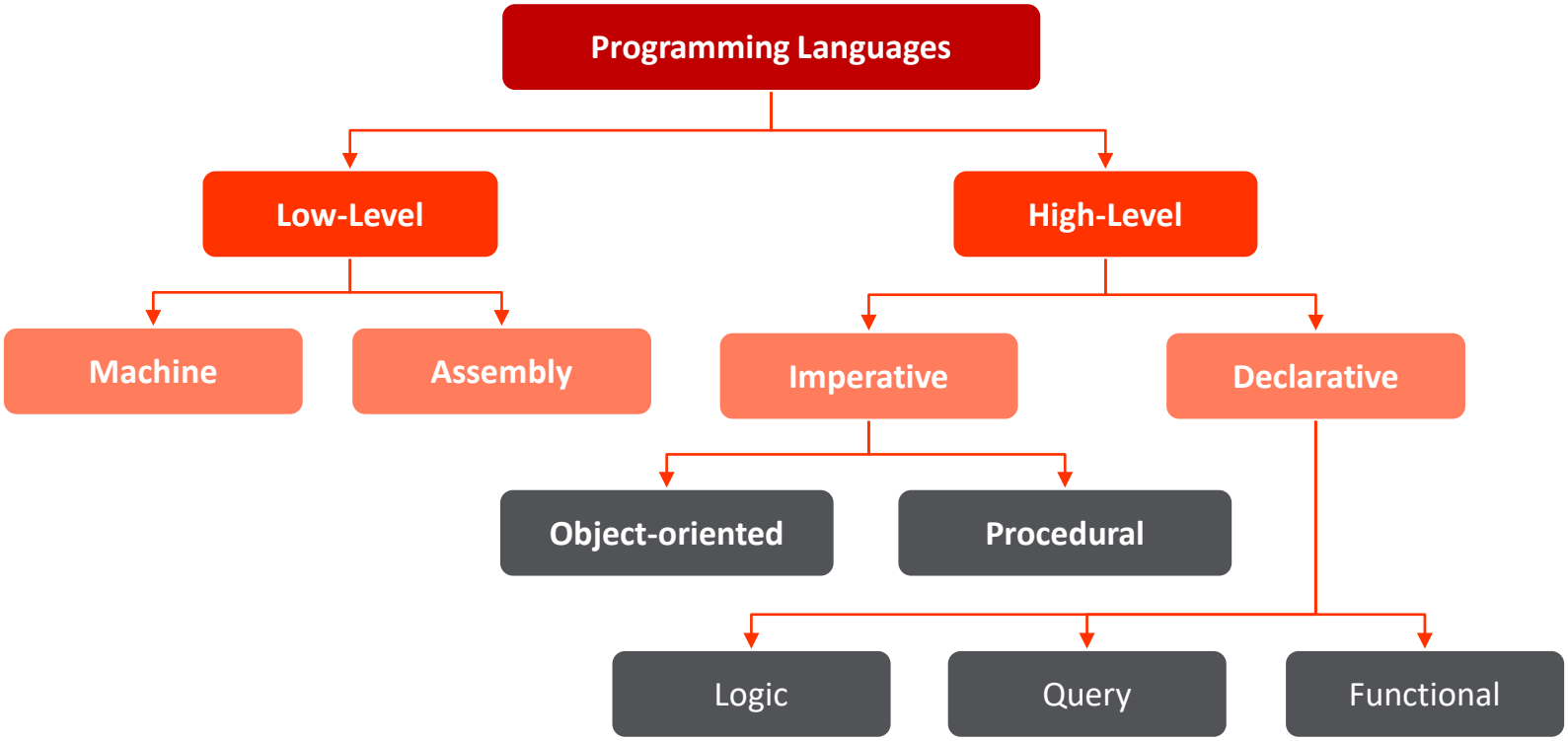
An __algorithm__ is a __procedure__ or __formula__ for solving a problem, based on conducting a sequence of specified actions. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

- 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

Classification of Programming Languages

Thousands of programming languages exist today in the market, with various purposes to fulfill. Some programming languages provide less or no abstraction from the hardware, whereas some other languages provide a higher level abstraction. To separate programming languages on the basis of level of abstraction from hardware, they are classified into various categories; however, there are two main classifications; Low Level and High-Level languages.



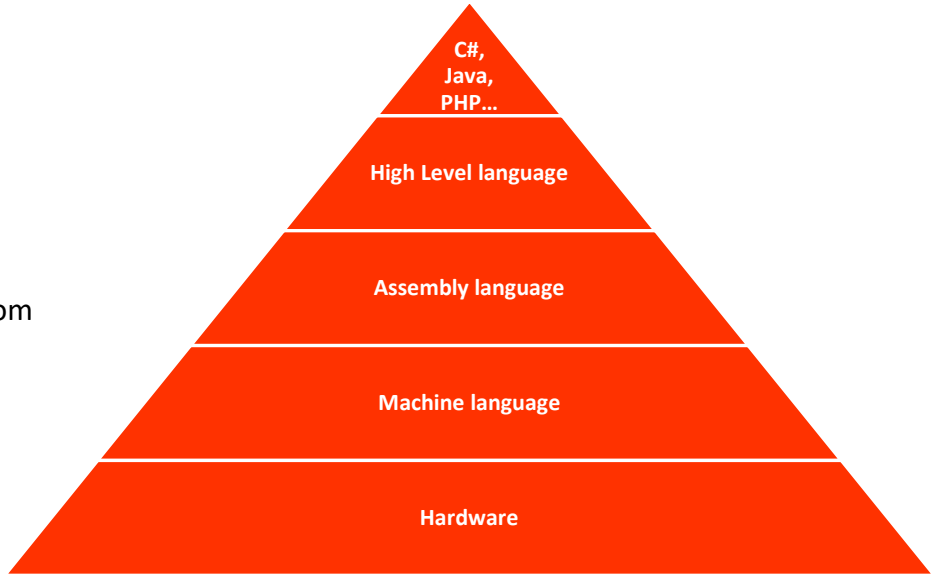
Source: https://codeforwin.org/2017/05/programming-languages-classification.html

Programming Languages

The **three major families** of languages are:

- Machine languages
- Assembly languages
- High-Level languages

The **abstraction** level of programming languages from hardware is what distinguishes programming languages, with machine languages providing no abstraction, assembly languages providing less abstraction from the hardware, whereas high level languages provide a higher level of abstraction.



Machine Language

Definition:

A computer programming languages consisting of binary or hexadecimal instructions which a computer can respond to directly and execute directly through it's CPU.

- Comprised of 1s and 0s
- The "native" language of a computer
- Difficult to program – one misplaced 1 or 0 will cause the program to fail

- **Code example:**

```
1110100010101
111010101110
10111010110100
10100011110111
```

Source: https://codeforwin.org/2017/05/programming-languages-classification.html

Compiled vs. Interpreted Languages

Every program is a set of instructions, whether it's to add two numbers or send a request over the internet. Compilers and interpreters take human-readable code and convert it to computer-readable machine code. In a compiled language, the target machine directly translates the program. In an interpreted language, the source code is not directly translated by the target machine. Instead, a *different* program, aka the interpreter, reads and executes the code.

Definition:
Compiled languages *are converted directly into machine code that the processor can execute.*

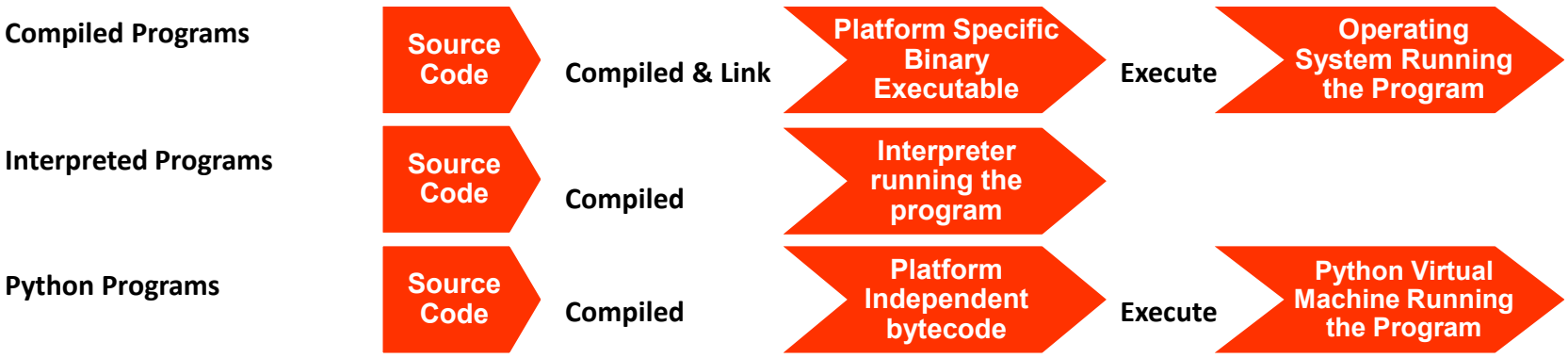
- Tend to be faster and more efficient to execute than interpreted languages
- Give the developer more control over hardware aspects, like memory management and CPU usage.
- Need a "build" step - they need to be manually compiled first. You need to "rebuild" the program every time you need to make a change.

Examples of pure compiled languages are C, C++, Erlang, Haskell, Rust, and Go.

Definition:
Interpreters will run through a program line by line and execute each command.

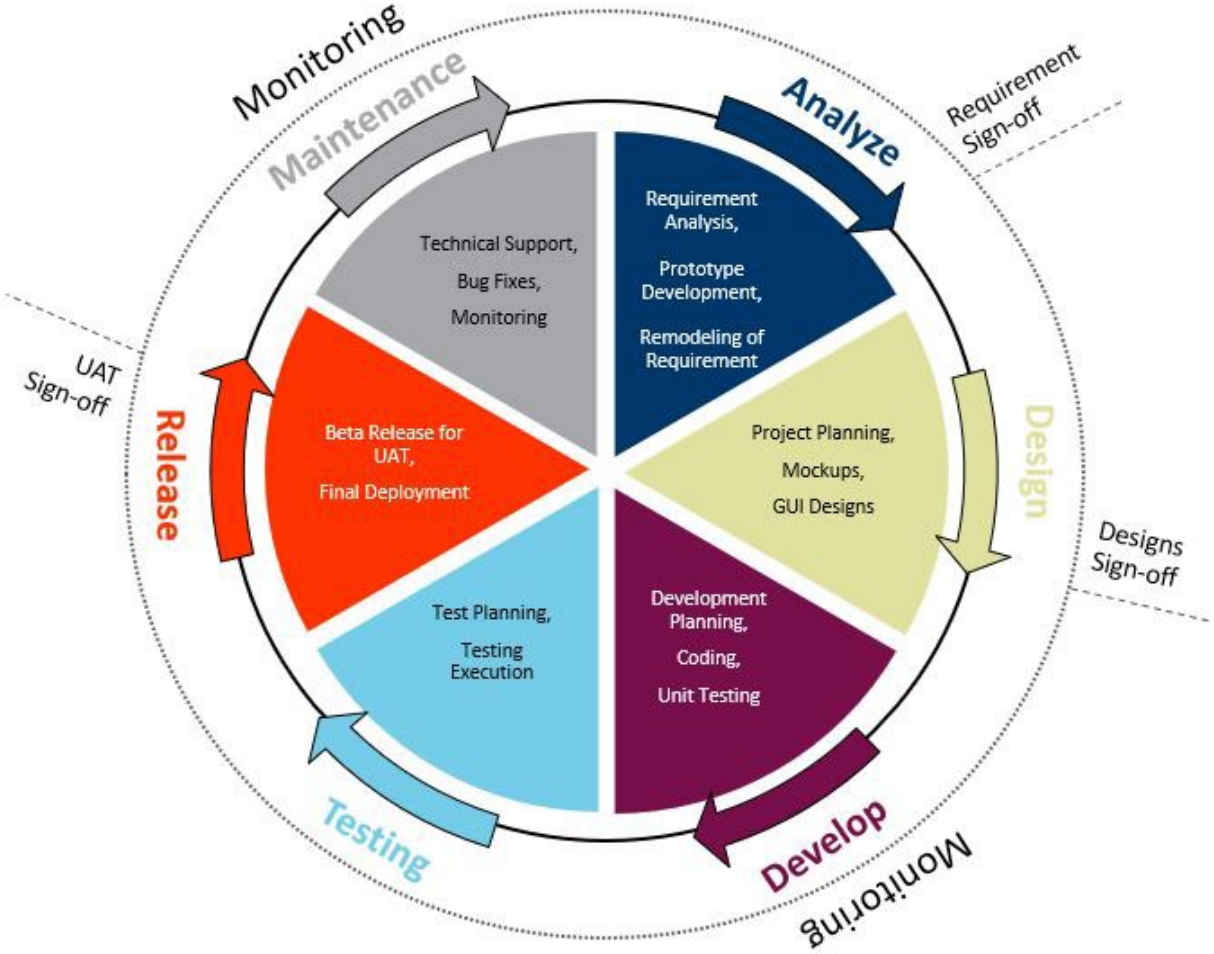
- Interpreted languages were once known to be significantly slower than compiled languages. But, with the development of just-in-time compilation, that gap is shrinking.

Examples of common interpreted languages are PHP, Ruby, Python, and JavaScript.



Source: https://guide.freecodecamp.org/computer-science/compiled-versus-interpreted-languages/

Application Development Process



Based on 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

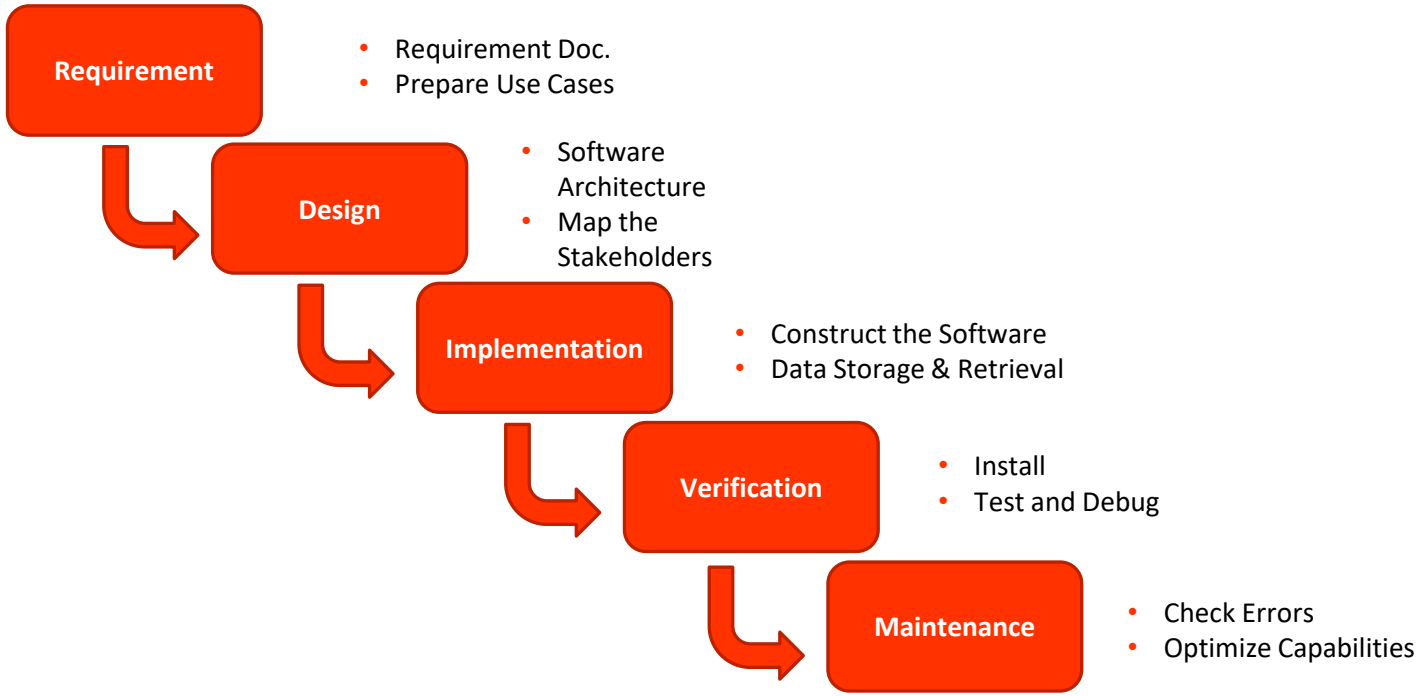
Waterfall

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

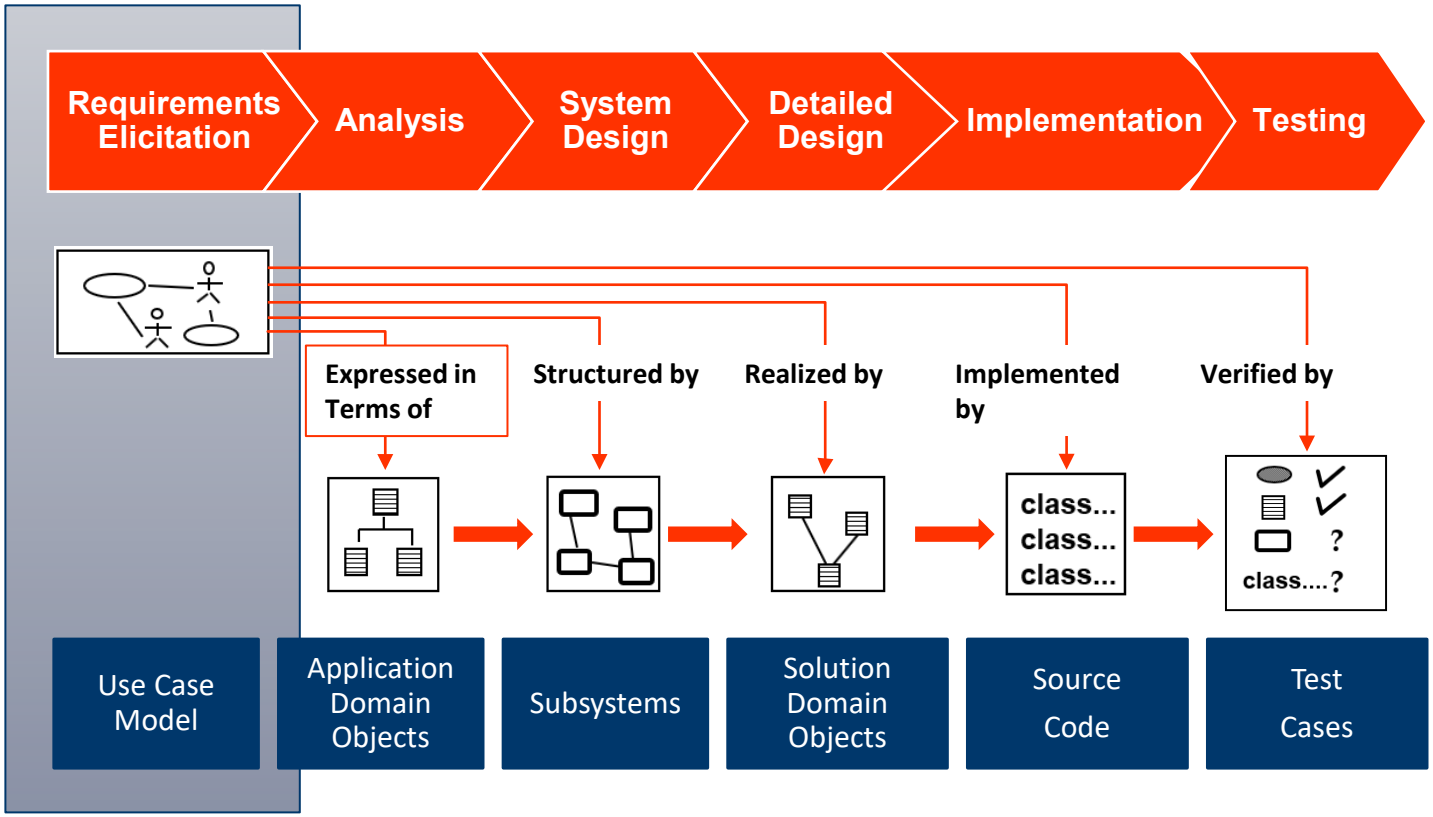
- **The result of each phase is one or more documents which are approved ("signed off")**

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**

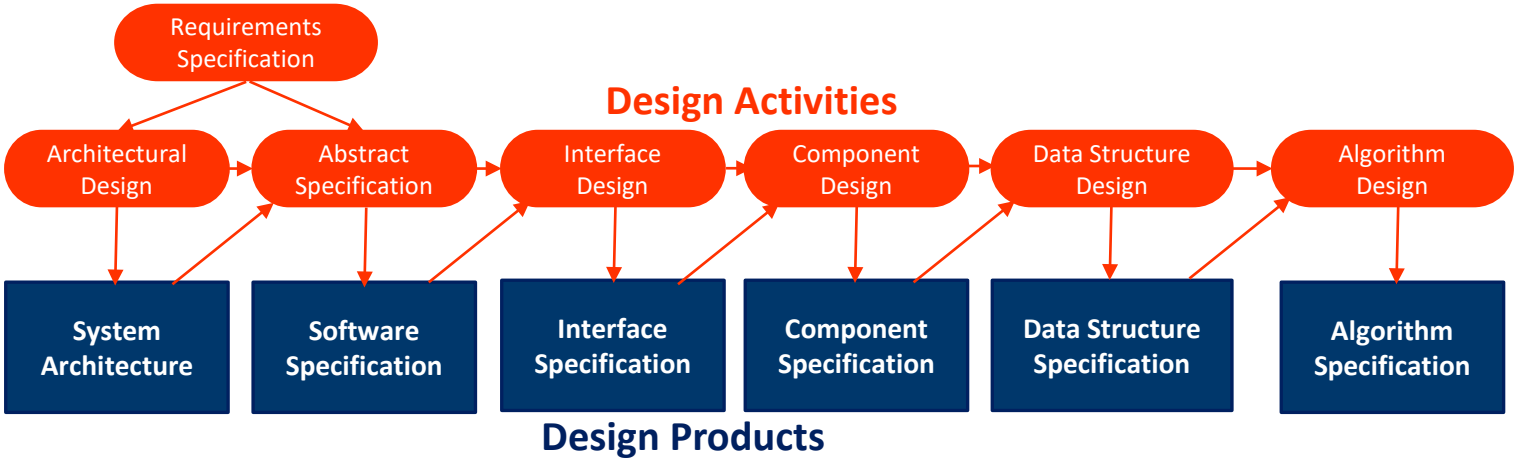
Source: https://iansommerville.com/software-engineering-book/

Software Development Lifecycle Activities



Adapted from: https://iansommerville.com/software-engineering-book/

The Software Design Process



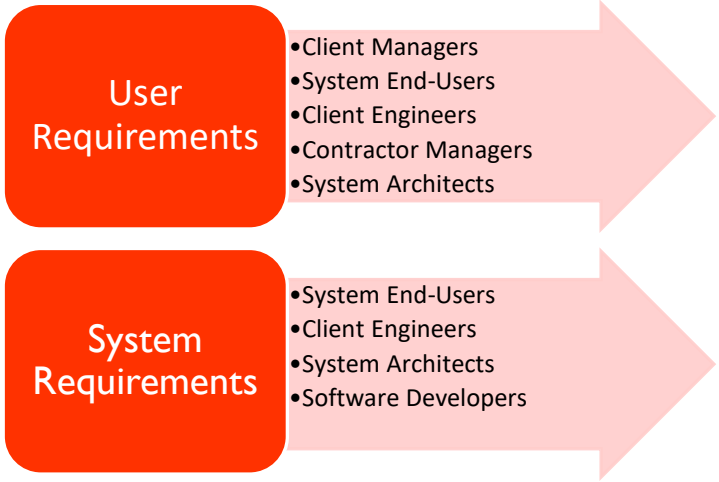
Requirements Capturing (1/5)

Definition:

Determine the needs or conditions to meet taking account of the possibly conflicting requirements of the various stakeholders, analyzing, documenting, validating and managing software or system requirements

In systems engineering and software engineering, **requirements analysis** encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product or project, taking account of the possibly conflicting requirements of the various stakeholders, *analyzing, documenting, validating and managing* software or system requirements, and is vital for the success or failure of a software project.

The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.



Sources: https://iansommerville.com/software-engineering-book/
https://en.wikipedia.org/wiki/Requirements_analysis

Requirements Capturing (2/5)

Customer Requirements:

Definition:

Statements of fact and assumptions that define the expectations of the system in terms of mission objectives, environment, constraints, and measures of effectiveness and suitability (MOE/MOS). The customers are those that perform the eight primary functions of systems engineering, with special emphasis on the operator as the key customer.

Operational Requirements:

Definition:

Will define the basic need and, at a minimum, answer the questions posed in the following listing:

- Operational distribution or deployment: Where will the system be used?
- Mission profile or scenario: How will the system accomplish its mission objective?
- Performance and related parameters: What are the critical system parameters to accomplish the mission?
- Utilization environments: How are the various system components to be used?
- Effectiveness requirements: How effective or efficient must the system be in performing its mission?
- Operational life cycle: How long will the system be in use by the user?
- Environment: What environments will the system be expected to operate in an effective manner?

Source: https://en.wikipedia.org/wiki/Requirements_analysis#Types_of_requirements

Requirements Capturing (3/5)

- **Architectural requirements:** explain what has to be done by identifying the necessary systems architecture of a system.
- **Structural requirements:** explain what has to be done by identifying the necessary structure of a system.
- **Behavioral requirements:** explain what has to be done by identifying the necessary behavior of a system.
- **Functional requirements:** explain what has to be done by identifying the necessary task, action or activity that must be accomplished. Functional requirements analysis will be used as the top-level functions for functional analysis.
- **Non-functional requirements:** requirements that specify criteria that can be used to judge the operation of a system, rather than specific behaviors.
- **Core functionality** and **ancillary functionality requirements:** Murali Chemuturi defined requirements into core functionality and ancillary functionality requirements. Core functionality requirements are those without fulfilling which the product cannot be useful at all. Ancillary functionality requirements are those that are supportive to core functionality. The product can continue to work even if some or all of the ancillary functionality requirements are fulfilled but with some side effects. Security, safety, user friendliness and so on are examples of ancillary functionality requirements.

Source: https://en.wikipedia.org/wiki/Requirements_analysis#Types_of_requirements

Requirements Capturing (4/5)

- **Performance requirements**
  The extent to which a mission or function must be executed; generally measured in terms of quantity, quality, coverage, timeliness or readiness. During requirements analysis, performance (how well does it have to be done) requirements will be interactively developed across all identified functions based on system life cycle factors; and characterized in terms of the degree of certainty in their estimate, the degree of criticality to system success, and their relationship to other requirements.

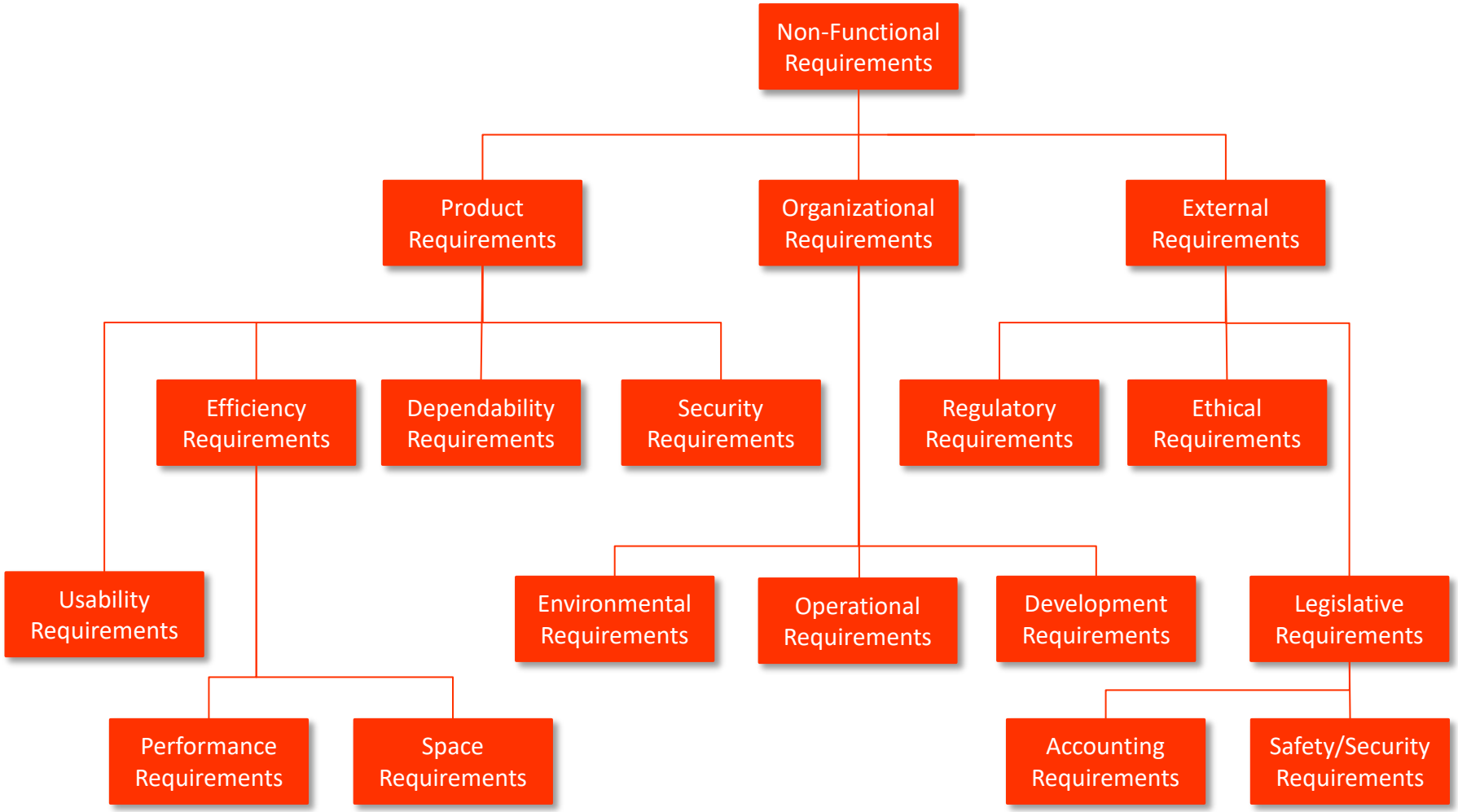
- **Design requirements**
  The "build to", "code to", and "buy to" requirements for products and "how to execute" requirements for processes expressed in technical data packages and technical manuals.

- **Derived requirements**
  Requirements that are implied or transformed from higher-level requirement. For example, a requirement for long range or high speed may result in a design requirement for low weight.

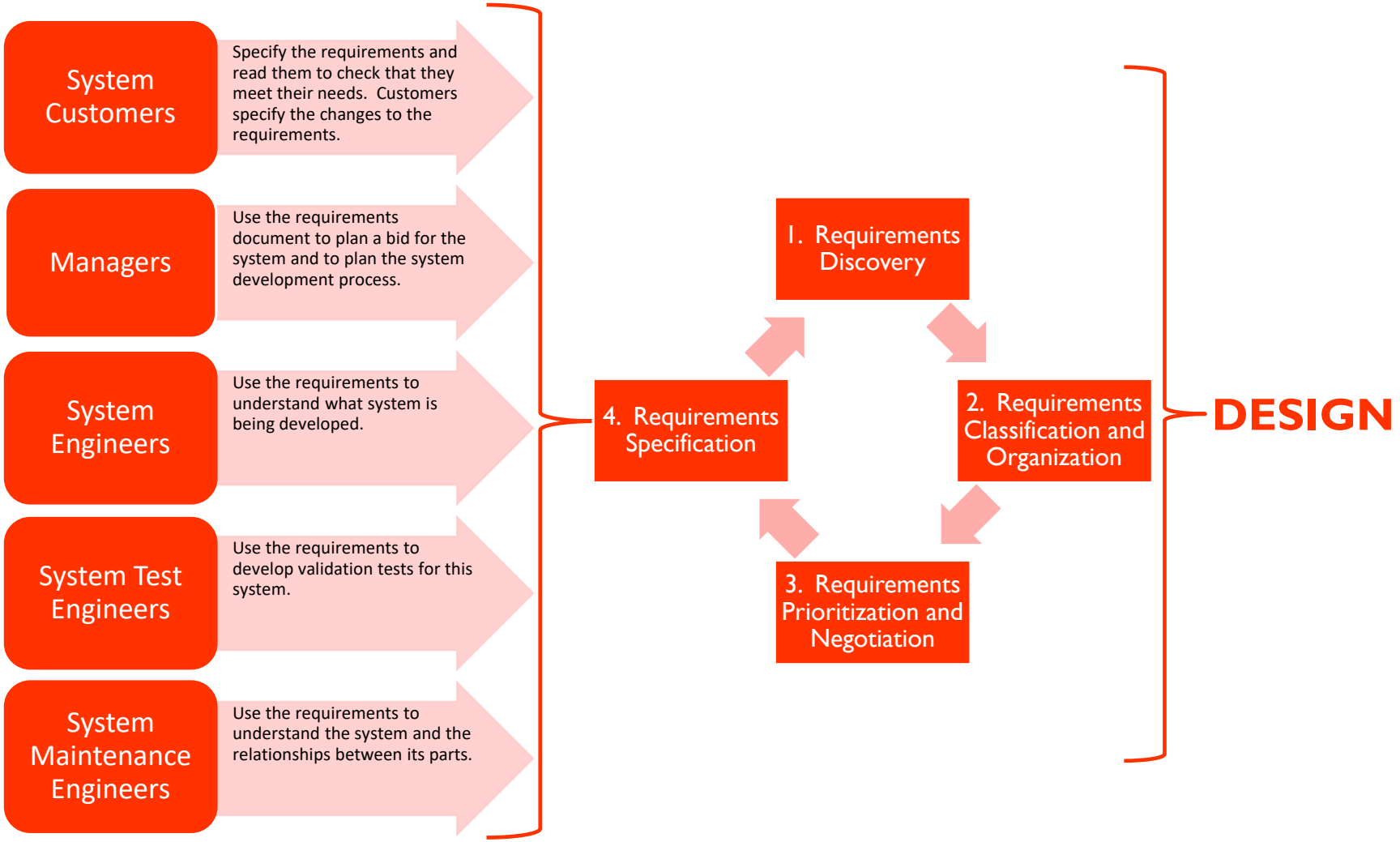
- **Allocated requirements**
  A requirement that is established by dividing or otherwise allocating a high-level requirement into multiple lower-level requirements. Example: A 100-pound item that consists of two subsystems might result in weight requirements of 70 pounds and 30 pounds for the two lower-level items.

Source: https://en.wikipedia.org/wiki/Requirements_analysis#Types_of_requirements

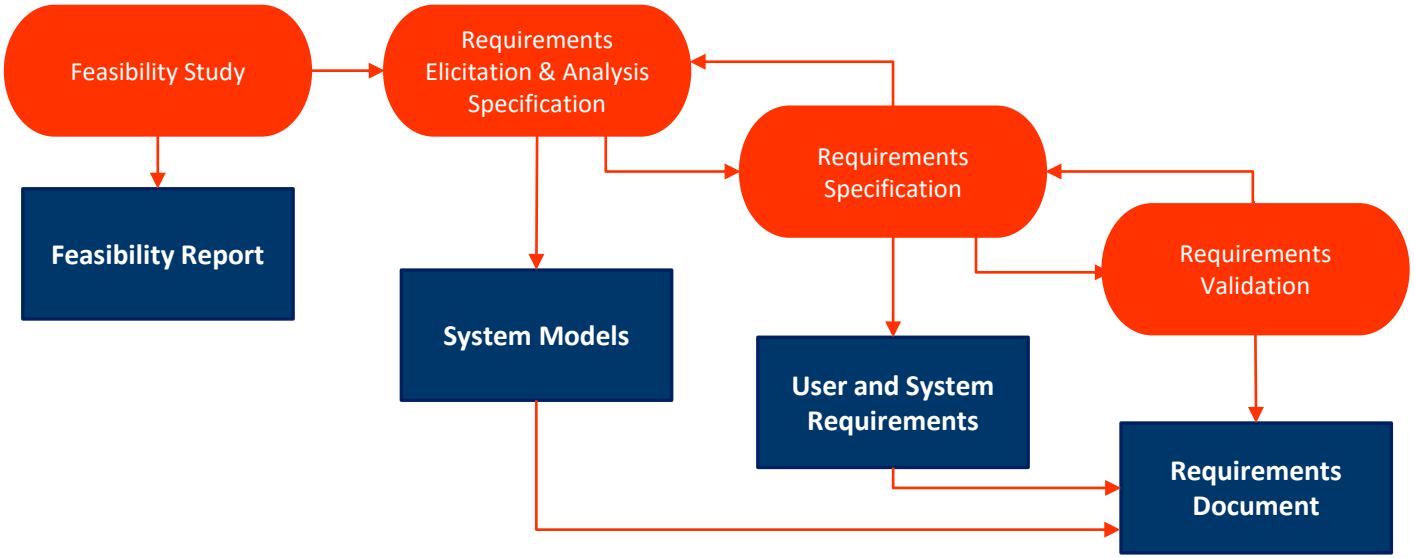
Requirements Capturing (5/5)



From Requirements Capturing to Software Design



The Requirements Engineering Process



5 Principles to Good Requirements

1. **Communicate Input to Design**
   - What are we solving?
   - Why is this function important?
   - Clarity to Cross-Functional Team

2. **Measurable & Testable**
   - Verification and Validation are Possible
   - Subjective Requirements cannot be Verified

3. **Requirements are Focused**
   - Audience for Requirement is known

4. **Provide Value to Development**
   - Based on Need: Answer WHY?

5. **Free of Specific Design Content**

Class Exercise | Discussion

Consider a simple game, like tic-tac-toe

- Draw a flowchart showing how the game works
- What are some basic requirements?

Sample Solution

- The game is played with 2 players, player A and Player B, using a board
- The board consists of 3x3 matrix
- Player A gets Os and Player B gets Xs
- Players to play alternatively, one turn each
- Draw a players option using an X/O in one of the matrix boxes
- Boxes cannot be reused
- If 3 consecutive boxes at any line direction are identical then this Player wins

Unified Modelling Language (UML)

Definition:

A standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

During a software development project, different people (i.e., designers, coders, testers, the customer, auditors) are interested in different aspects of the system, and each of them require a different level of detail. There are THIRTEEN (13) different type of diagrams; **Structural diagrams** show the static structure of the system while **Behavior diagrams** show the dynamic behavior of the objects in the system.

- The most popular diagrams are:
  - **Use Case Diagram** (behavior)
  - **Class Diagram** (structural)
  - **Sequence Diagram** (behavior)
  - **State Transition Diagram** (behavior)

Source: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

Use Case Diagram

Definition:

A use case diagram is "a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved".

Consider the case below:

This would become:

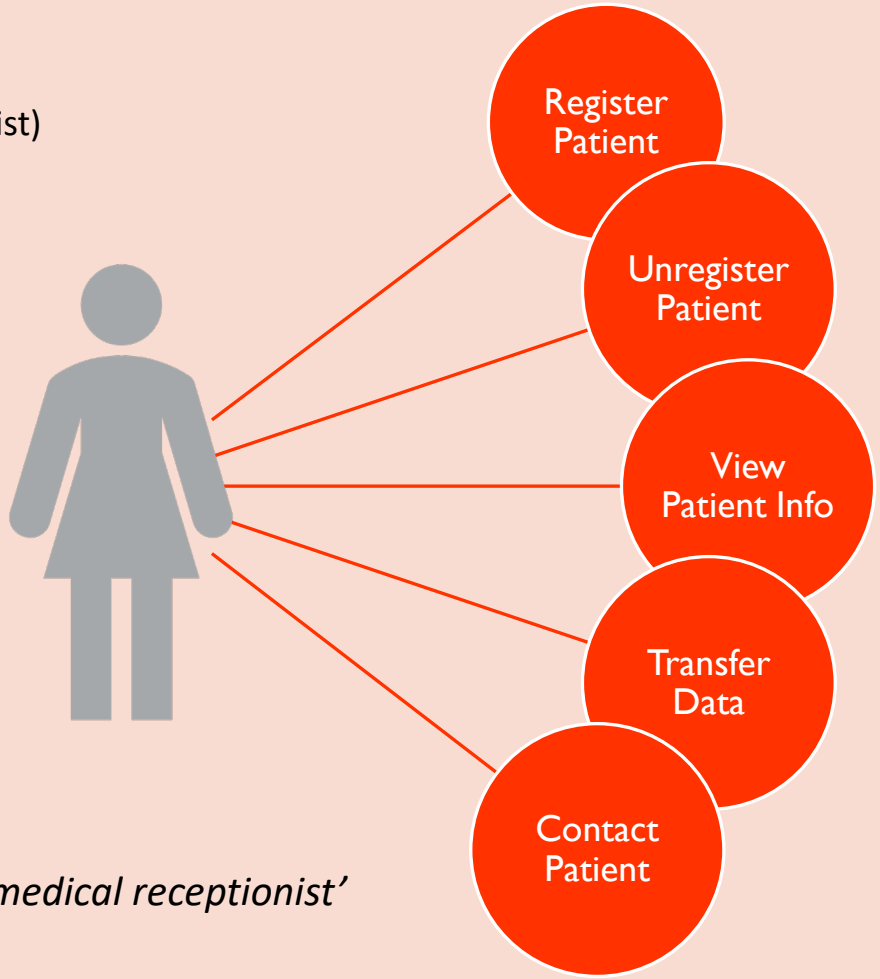
Transfer-Data Use Case Diagram Example

Adapted from: https://iansommerville.com/software-engineering-book/

Class Exercise | Discussion

Using the previous diagram as an example, draw a Use Case Diagram showing:

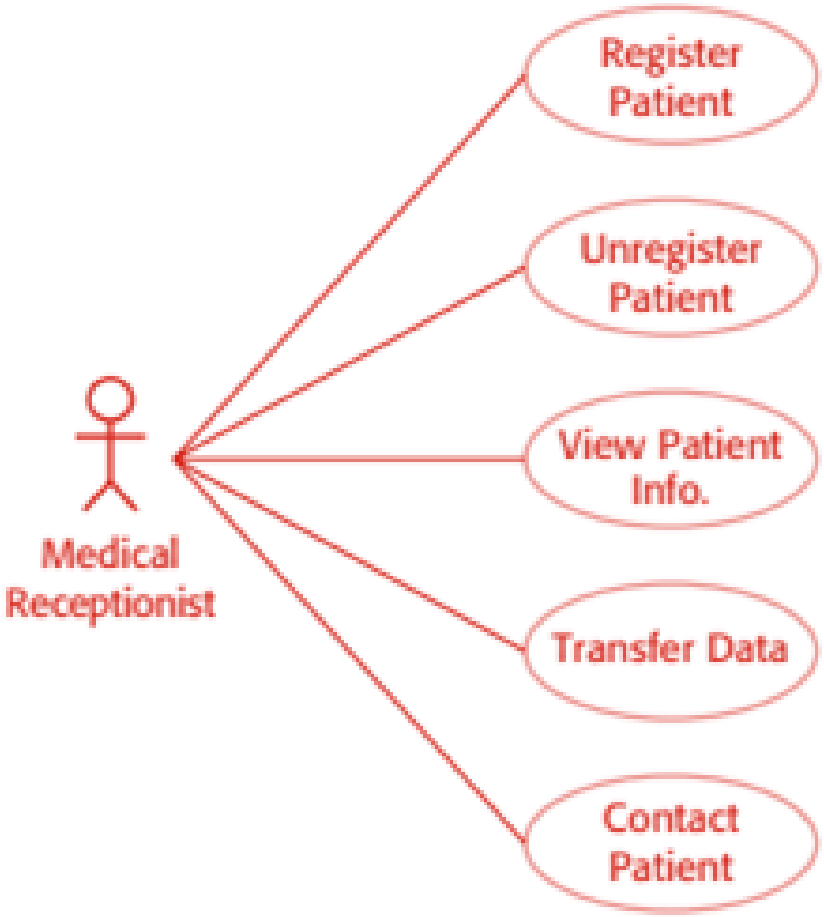
- A medical receptionist
- 4 Use Cases (any from the below list)
  - Register patient
  - Unregister Patient
  - View Patient Info
  - Transfer Data
  - Contact Patient



Use cases involving the role of 'medical receptionist'

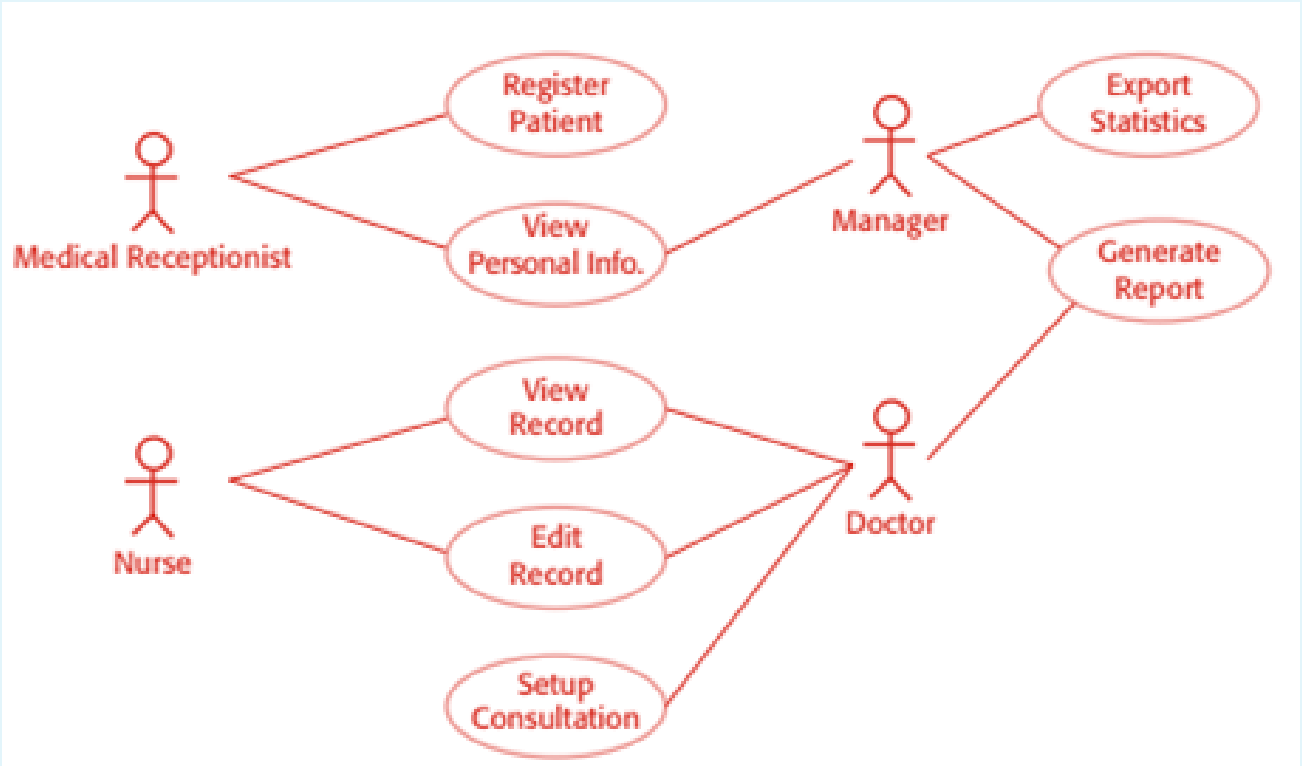
Adapted from: https://iansommerville.com/software-engineering-book/

Use Case Diagram | Discussion



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

Use Case Diagram | Sample Solution



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

Class Diagram

Definition:

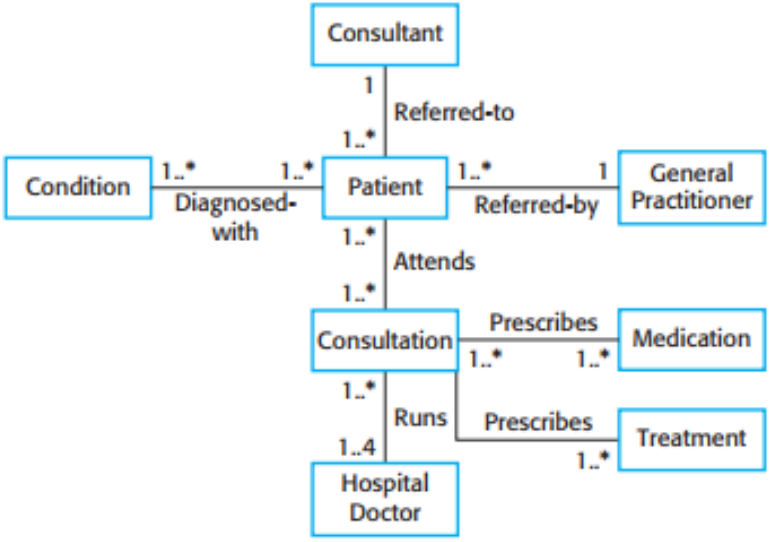
*A **class diagram** is "a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.*

In the diagram, **classes** are represented with boxes that contain three compartments:

1. The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
2. The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
3. The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

A class with three compartments.

In the design of a system, a number of classes are identified and grouped together in a class diagram that helps to determine the static relations between them. With detailed modeling, the classes of the conceptual design are often split into a number of subclasses.



Classes and Associations Example

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

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