CS 2212

Introduction to Software Engineering

Chapter 1

Key Principles – Software Life Cycle - Process

Copyright Notice

This Presentation and Its Associated Video Contain Copyright Material. This Presentation and Its Associated Video Are Provided for Teaching Purposes of CS2212 and Should Not Be Posted In Any External Site, Or Shared with. This Presentation and Its Associated Video Are Provided For Instructional Personal Use Only, and Only For The Purposes of the CS2212 Course.

Part 1

Key Software Engineering Principles

Truth can only be found in one place: the code.

- Robert C. Martin

Learning Objectives in this Part

1. To define Software Engineering as an engineering discipline
2. To discuss and disambiguate certain myths about software development

What is Software Engineering ?

repeatable traceable. orginized.

how good wire

1. It is an area of Computer Science which relates to techniques, methods, practices and tools for the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

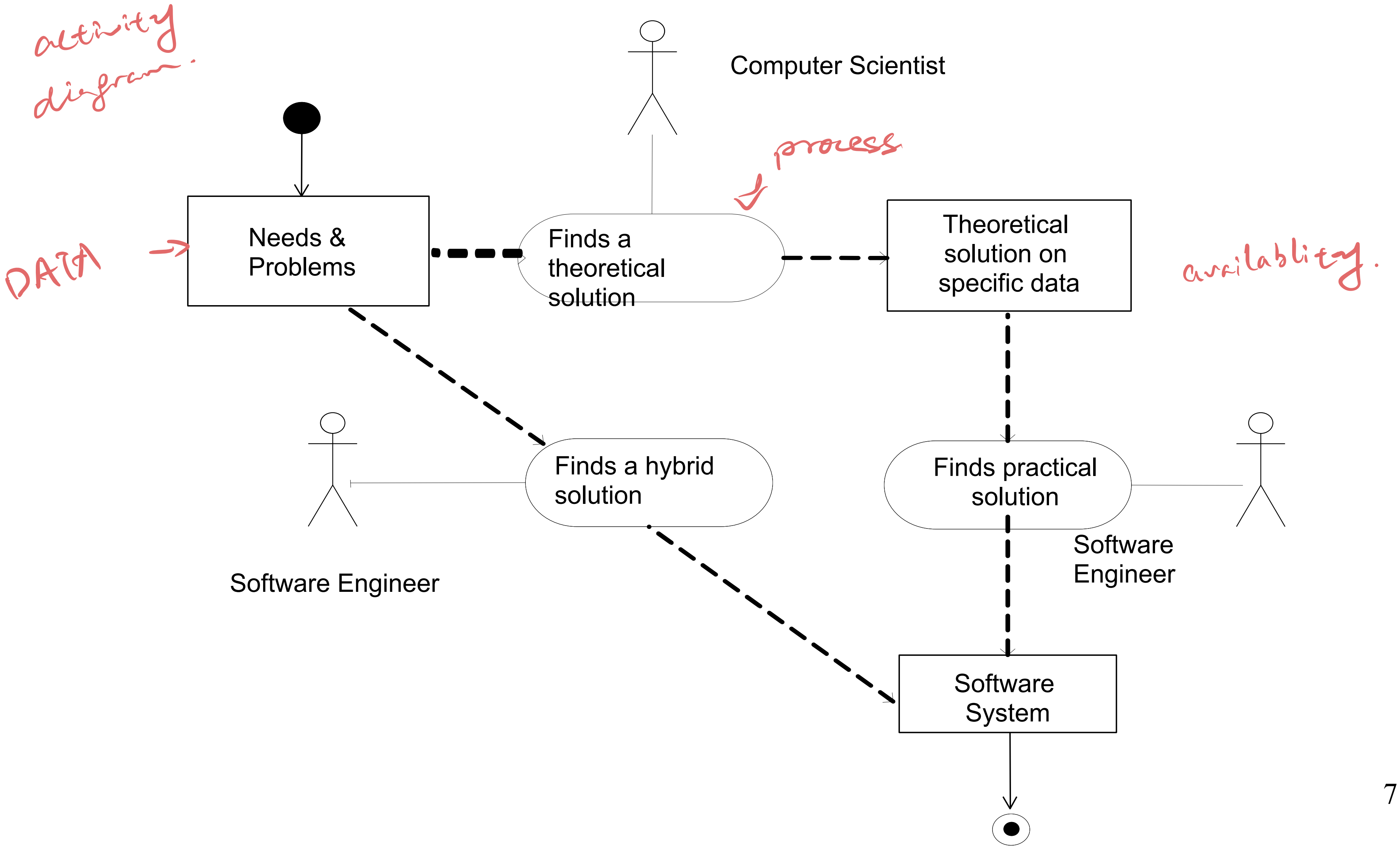
2. The study of approaches as in (1) in order to develop high quality products within the specified budget and time.

[IEEE Standard 610.12]

Distinct and Complementary Roles

- Computer Scientist
  - Aims for providing theorems and properties of algorithms, designing new programming languages, defining new information modeling formalisms etc.
  - A Computer Scientist usually has unlimited time to complete his/her task
- Engineer
  - Designs and oversees the implementation of a specific technical problem and project in a specific application domain
  - An engineer uses computers, programming languages, techniques, and tools
- Software Engineer
  - Usually works on the design of software systems and oversees the implementation of a specific technical software projects in different application domains
  - Has 3 months to design a solution ….
  - …while user requirements and available technology constantly change … ☹

Routes for Software Systems



Your Turn

- Why do we need software engineering practices? Isn't programing just enough to build software?

  Because we must consider the capability, the hardware.

- What are the common roles of a scientist, a computer scientist, and a software engineer with respect to developing software systems?

  scientists : design engineer : practice.

- What are the artifacts produced as a result of a software engineering project?

  Softwares.

- Read the content of the following sites:
  - https://en.wikipedia.org/wiki/Software_engineering
  - https://www.ibm.com/topics/software-development
  - https://www.d.umn.edu/~gshute/softeng/principles.html

Software Myths

- Myth #1:

  We already have a book that's full of standard and procedures for building software. Won't that provide my people with everything they need to know?

  how people design thing
  implement.

- Reality:

  ?

  Is this book used? Are people aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Chances are, the answers to these questions is no.

Software Myths

- Myth #2:

  If we get behind schedule, we can add more programmers and catch up (sometimes called the "Mongolian Horde" technique).

- Reality:

  Adding people to a late software project makes it later. As new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added, but only in a planned and well coordinated fashion.

education managing
communiation may break after 7 people.

Software Myths

- Myth #3:

  If I decide to outsource a software project to a third party, I can just relax and let that firm build it.

- Reality:

  If an organization does not how to manage and control software projects internally, it will invariably struggle when it outsources software projects too.

Software Myths

- Myth #4:

  A general statement of objectives is sufficient to begin writing programs – we can fill in the details later.

  actually, these will never begin if you don't understand the detail

- Reality:

  Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster. Effective and continuous communication between customer and developer is necessary.

Software Myths

- Myth #5:

  Software requirements continually change, but change can be easily accommodated because software is flexible. ✗ ???

- Reality:

  While software requirements can change, the impact of the change varies with the time at which it is introduced. The later changes are introduced, the more problematic they are, as great portions of design and/or code might need to be discarded and redone.

Software Myths

- Myth #6:

  Once we write the program and get it to work our job is done.

- Reality:

  Someone once said that “the sooner you begin ‘writing code’ the longer it’ll take you to get done”. Industry data indicate that between 60 and 80 percent of all effort expended on software occurs after delivery to the customer for the first time.

Software Myths

- Myth #7:

  Until I get the program running, I have no way of assessing its quality.

- Reality:

  One of the more effective software quality assurance mechanisms, the technical review, can be applied from the start of a project. Do not wait until the end to start assessing quality!

Software Myths

- Myth #8:

  The only deliverable work product for a successful project is the working program.

- Reality:

  A working program is only part of the solution. A variety of work products (e.g. models, documents, plans) provide a foundation for successful engineering and guidance for software support.

Software Myths

- Myth #9:

  Software engineering will make us create voluminous and unnecessary documentation and work and will invariably slow us down.

ensure product has higher quality

- Reality:

  Software engineering is about creating a quality product. (Yes, some documentation and extra effort is required for this.) Better quality leads to reduced rework, and this reduced rework results in faster delivery times than possible otherwise.

Your Turn

- Why the existence of standards (e.g. ISO, IEEE) are not enough when developing software systems?

  Because it requires HOW people design

- A project runs late. Why adding more developers won't fix the problem?

  Communication Education Managment

- From the perspective of a client who commissions a new software project, what are the issues to be aware when outsourcing the development to a third party?

  Make the third party know what should the outcome be used for.

- Can we assess the quality related of a software system being built before we complete coding the whole system?

  No

- Read the content of the following sites:
  - https://www.softwaretestinghelp.com/what-is-sei-cmm-iso-ieee-ansi-will-it-help/
  - https://stackoverflow.blog/2019/10/01/pitfalls-avoid-outsourcing-software-development/
  - https://en.wikipedia.org/wiki/Software_development_effort_estimation
-

Part 2

Dealing with Complexity

There are two ways of constructing a software design. ①One way is to make it so simple that there are obviously no deficiencies. The other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult. —C.A.R. Hoare

Simple is beautiful.

Learning Objectives in this Part

1. To obtain a first look on the guiding principles and approaches software engineers use to handle complexity towards developing large scale systems

Dealing with Complexity

1. Abstraction : Perform abstraction when modeling artifas.
   - Modeling and Object Orientation helps

     focus on what's important
2. Decomposition Make complex problem into smaller problem and solved seperately.
   - Cohesion, Coupling, Modularity

     commuting in a clean way.
3. Hierarchy :
   - Specializations, Generalizations

     custom your own solution with current framework and codes,

Principles that Guide Practice

- **Principle #1**. **Divide and conquer**. Stated in a more technical manner, analysis and design should always emphasize separation of concerns (SoC), subdividing large problems into a collection of elements (or concerns) that are easier to address.

  ↓ Different modules for different parts.

- **Principle #2**. **Understand the use of abstraction**. At its core, an abstraction is a simplification of some complex element of a system used to communicate meaning in a single phrase. e.g. interface, class.

  Finishing the problem in a simple, concise and sound manner

- **Principle #3**. Strive for consistency. A familiar context makes software easier to specify, built, test, and use.

  Using common language.
  e.g. variable name

Principles that Guide Practice

How we design communication?

- **Principle #4**. Focus on the transfer of information. Software is all about information transfer, and there are risks of error, omission, or ambiguity. Pay special attention to the analysis, design, construction, and testing of interfaces as information flows across these junctures.

- **Principle #5**. Build software that exhibits effective modularity. Separation of Concerns (Principle #1) establishes a philosophy for software. Modularity provides a mechanism for realizing this philosophy.

Make composition clean.
Each module works one specific thing.
One module transfer only information to another module.

Principles that Guide Practice

apply patterns to problems.

- **Principle #6**. Look for patterns. Brad Appleton suggests that: "The goal of patterns within the software community is to create a body of literature to help software developers resolve recurring problems encountered throughout all of software development."

- △ **Principle #7**. When possible, represent the problem and its solution from a number of **different perspectives**. Taking in account of others.

Many users have many perspectives, programmers have to satisfy all of them.

- **Principle #8**. Remember that someone will maintain the software. Maintenance activities can be facilitated if solid software engineering practice is applied throughout the software process.

1. Corractive : Found the problem.
2. Adative : Upgrade the system.
3. Perfective : Add new function to the system.

Part 3

Software Applications

Inside every large program is a small program struggling to get out.

—Tony Hoare

Learning Objectives in this Part

To discuss types of software applications which currently dominate the market and pose development challenges

Software Applications

e.g: Operating system, Compiler,

- System software : Allow for infrastructure to operate.
- Application software : Solving problem in specific domain.
- Engineering/scientific software : Software that calculate / analyze data
- Embedded software : Control electronic devices.
- Product-line software : Involves different product lines edition
- Web/mobile applications : Allowing clients can communicate via end application
- AI software (robotics, neural nets, etc.)

  : finish complex tasks.

The Changing Nature of Software

- While we still have to deal with legacy software, a lot has changed and evolved over the years
- Some categories now dominate the software landscape that were in their infancy little more than a decade ago:
  - Web apps
  - Mobile apps
  - Cloud computing

Web Apps

- Modern web apps are often much more than hypertext files with accompanying graphics and other media content, though those elements are still a key aspect of web apps
- They are augmented with tools to give interactive computing capability and can be integrated with massive corporate databases and sophisticated business applications
- The aesthetic nature of the content remains an important determinant of the quality of a web app

Web Apps

- Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, web app can evolve and be released continuously (referred to as ***continuous software engineering***)

- Although immediacy—the need to get software to market quickly—is a characteristic of many application domains, web apps often exhibit a time to market that can be a matter of a few days or weeks

- Because web apps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application and so security is an important issue limited resources

Mobile Apps

take advantage of sensors e.g. touchscreen.

- Reside on mobile platforms (for example iOS and Android) on devices like phones or tablets

constant change information.
to backend.

- Take advantage of unique interaction mechanisms provided by the mobile platform and devices (touch screens, accelerometer, vibration feedback, location via GPS, etc.)

- Often provide access to a combination of web-based resources and local device processing and storage capabilities

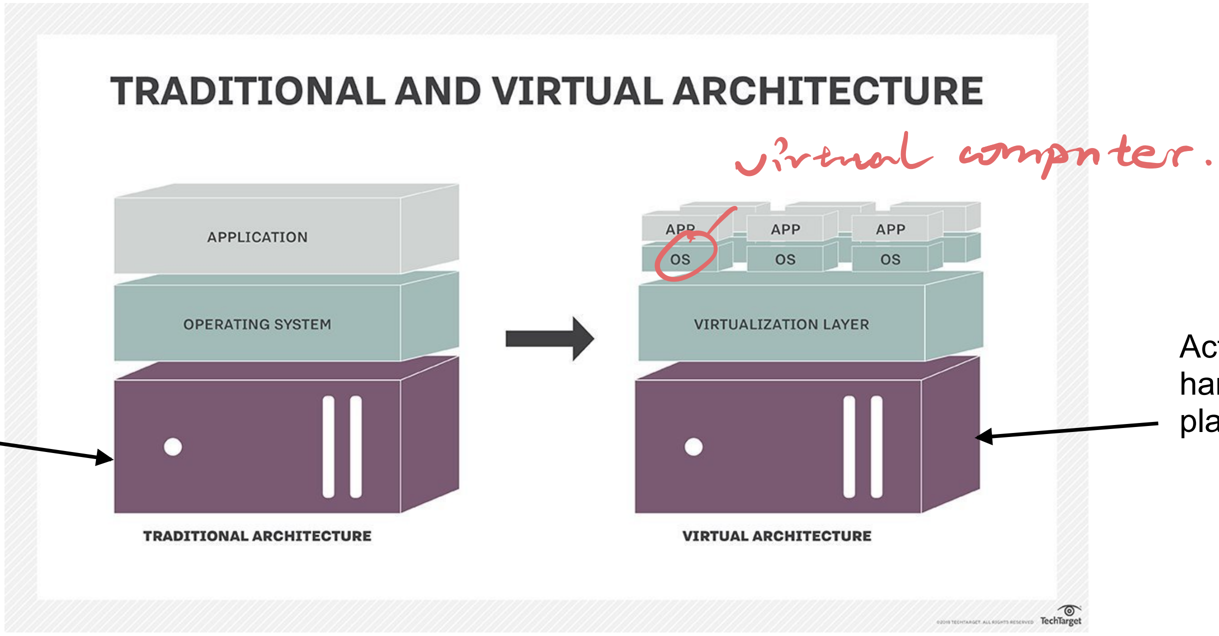
- Typically provide persistent storage capabilities on the device, but can also leverage remote cloud storage

Cloud Computing

- Cloud computing is a system-wide architectural paradigm which allows for the on-demand provision of resources (application software, platform, infrastructure) through virtualization technologies

- Virtualization technologies allow for the creation of a virtual (rather than actual) version of computer hardware platforms, storage devices, and computer network resources.

Actual hardware platform

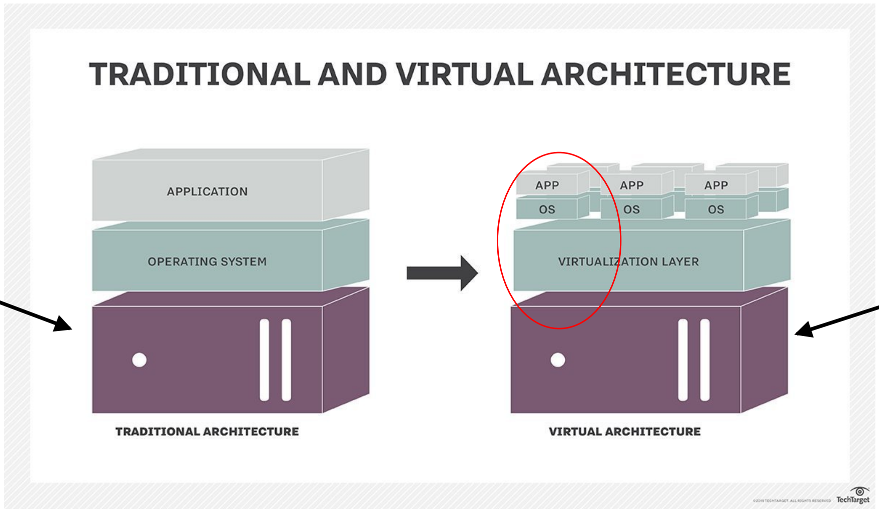


Actual hardware platform

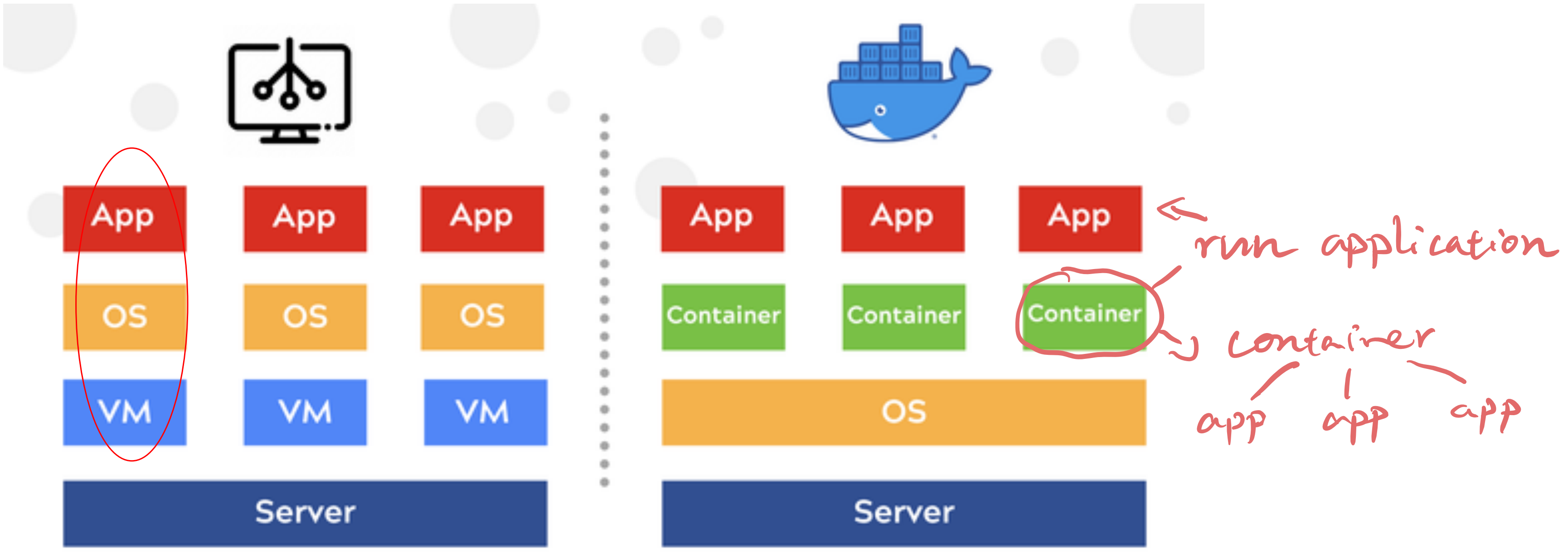
https://searchservervirtualization.techtarget.com/definition/virtualization

From Virtual Architectures to Microservices

Actual hardware platform

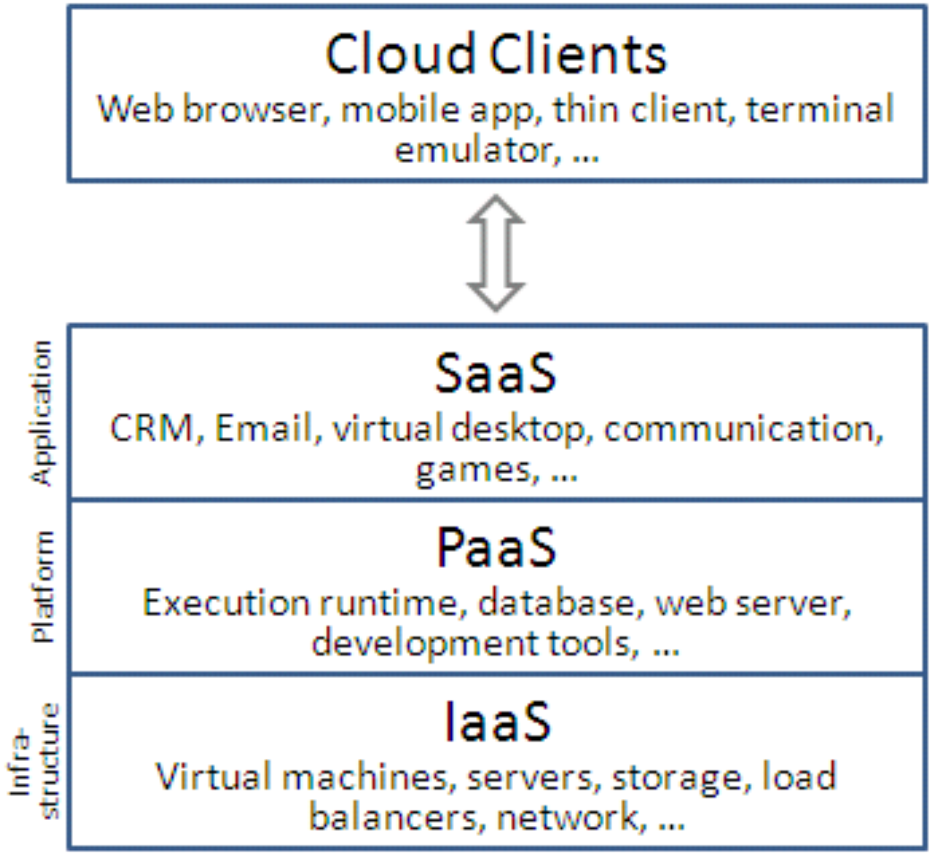


Actual hardware platform



Cloud Computing

- Cloud computing requires developing an architecture containing a collection of both frontend and backend services
  - Frontend services include the client devices and application software to allow access to what is hosted on the cloud
  - Backend services include servers, data storage, and server-resident applications residing in the cloud


Cloud Clients
Web browser, mobile app, thin client, terminal emulator, ...
Application
SaaS
CRM, Email, virtual desktop, communication, games, ...
Platform
PaaS
Execution runtime, database, web server, development tools, ...
Infra-structure
IaaS
Virtual machines, servers, storage, load balancers, network, ...

https://commons.wikimedia.org/w/index.php?curid=18327835

Your Turn

- What is modularity in software design and why is it important?

Make one module just work for one function

- What is the concept of abstraction in software engineering? Briefly discuss three examples of abstraction. Machine language → Assembly language

A technique for arranging complexity of computer system.

- With respect to software development, what are the major challenges presented by Web applications, Mobile applications, and Cloud applications?

Limited of backend.

- Read the content of the following sites:
  - https://en.wikipedia.org/wiki/Modular_programming
  - https://en.wikipedia.org/wiki/Abstraction_(computer_science)
  - https://www.tutorialspoint.com/software_architecture_design/hierarchical_architecture.htm
  - https://en.wikipedia.org/wiki/Component-based_software_engineering
  - https://en.wikipedia.org/wiki/Web_application
  - https://en.wikipedia.org/wiki/Cloud_computing
  - https://www.tutorialspoint.com/software_architecture_design/hierarchical_architecture.htm