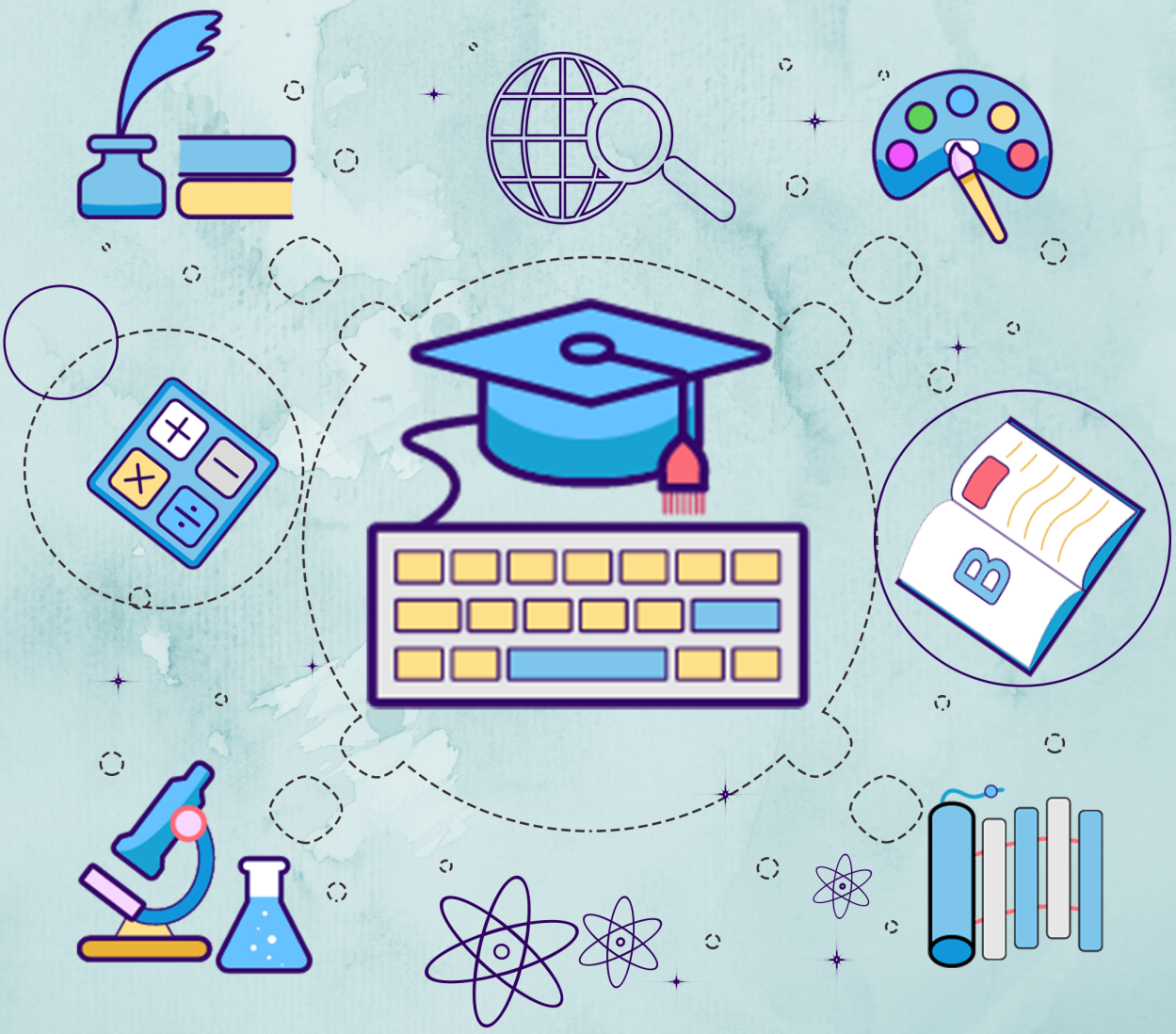
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KeralaNotes



SYLLABUS | STUDY MATERIALS | TEXTBOOK

PDF | SOLVED QUESTION PAPERS

www.keralanotes.com

KeralaNotes

KTU STUDY MATERIALS

MANAGEMENT OF SOFTWARE SYSTEMS

CST 309

Module 5

Related Link :

- KTU S5 STUDY MATERIALS
- KTU S5 NOTES
- KTU S5 SYLLABUS
- KTU S5 TEXTBOOK PDF
- KTU S5 PREVIOUS YEAR SOLVED QUESTION PAPER

Module V

Software Quality

Software quality can be defined as:

- *An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.*

Software quality can also be defined as how well the software works

For software, two kinds of quality may be encountered:

- Quality of design encompasses requirements, specifications, and the design of the system.

Which means characteristics of items used by the designer

- Quality of conformance is an issue focused primarily on implementation.
- User satisfaction = compliant product + good quality + delivery within budget and schedule

Quality Control

- The process in which several activities are conducted to maintain the quality of product.
- The activities are Inspection, review and testing
- The activities can be manual, automatic or semi-automatic

Quality Assurance:

Planned and systematic pattern of activities are done which are needed to provide high degree of reliability in software.

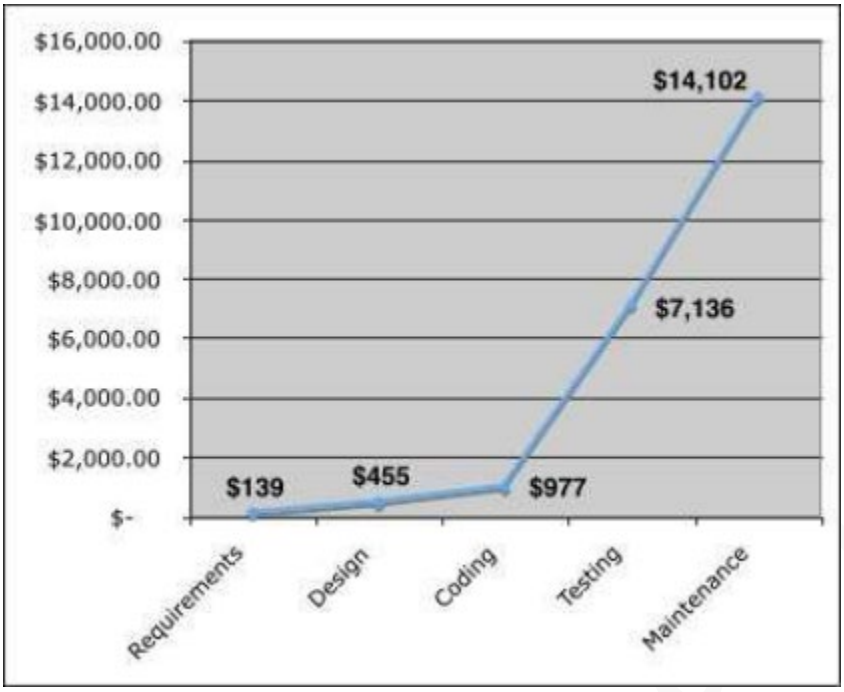
Cost of Quality:

Total cost required to obtain quality and conduct quality related activities

- 3 types of costs
    - Prevention cost – Quality planning, FTR, Test Equipment and trainings
    - Appraisal cost – Testing and Inspection
        - Inspection consist of in-process inspection and inter-process inspection

- in-process inspection means inspection done within the process
- inter-process inspection means inspection done between the process

- Failure cost
  - Internal
    - Internal failure cost means cost of defects before the software delivered to the customer
    - Rework and repair
    - Failure mode analysis
  - External
    - External failure cost means cost of defects after the software delivered to the customer
    - Complaint resolution
    - Product return and replacement
    - Helpline support
    - Warranty work
  - The relative costs to find and repair an error or defect increase dramatically as we go from prevention to detection to internal failure to external failure costs.



Software Quality Dilemma

The quality dilemma is basically two fold:

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- Spend too much time on quality measures and no one gets to buy/use it because it either costs too much, or is still being worked on.
- Either you missed the market window, or you simply exhausted all your resources.
- So people in industry try to get to that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so

much perfectionism and so much work that it would take too long or cost too much to complete.

Software Quality Assurance (SQA):

- The process which ensures developed software meets the quality specifications of standardized software
- This is done by software engineers group and SQA group

Software Quality Assurance (SQA) Tasks:

- Prepare an SQA plan for project
- Participate in development of the project software process description
- Review the current software engineering activities to verify the compliance of with the defined software process
- Audit the designated software work products to verify the compliance with those defined as part of software process
- Ensures that there are no deviations in software work
- Record any non-compliance and report to senior manager

Software process improvement (SPI)

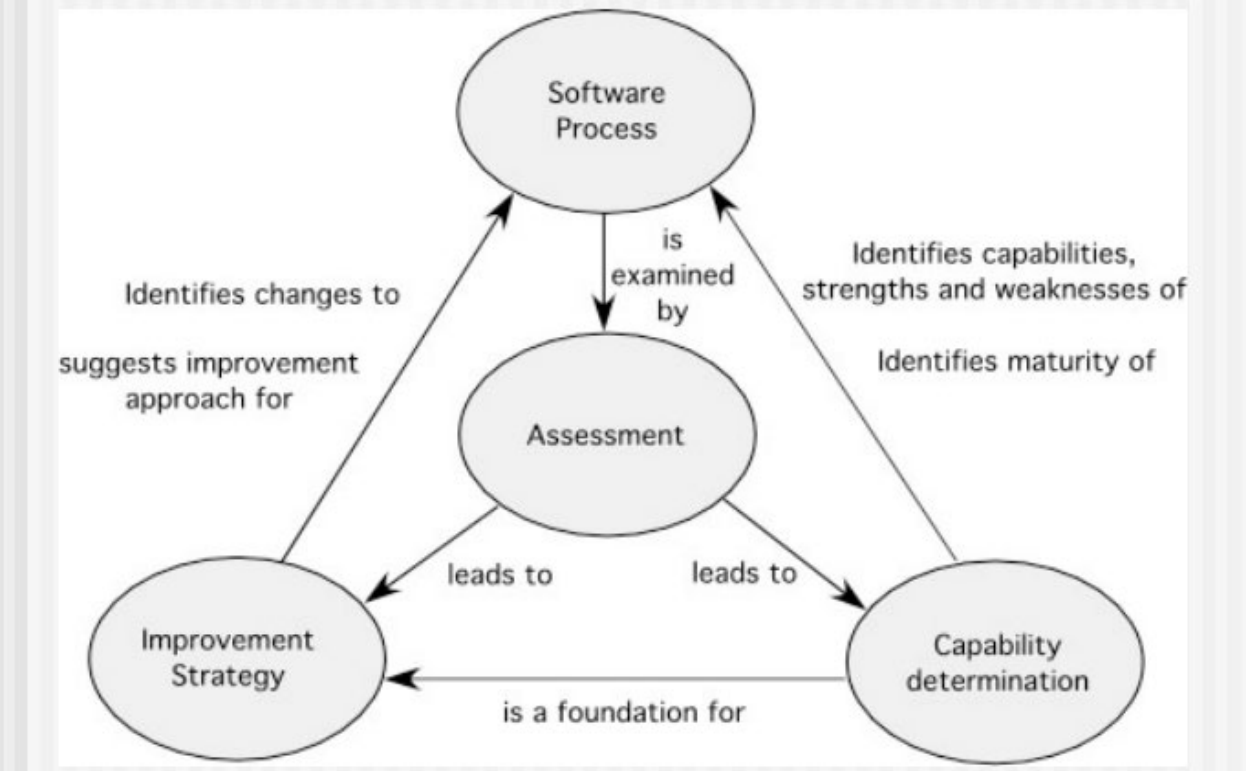
What is SPI?

- SPI implies that
  - elements of an effective software process can be defined in an effective manner
  - an existing organizational approach to software development can be assessed against those elements, and
  - A meaningful strategy for improvement can be defined.
- The SPI strategy transforms the existing approach to software development into something that is more focused, more repeatable, and more reliable (in terms of the quality of the product produced and the timeliness of delivery).

SPI Framework

- a set of characteristics that must be present if an effective software process is to be achieved
- a method for assessing whether those characteristics are present
- a mechanism for summarizing the results of any assessment, and
- a strategy for assisting a software organization in implementing those process characteristics that have been found to be weak or missing.
- An SPI framework assesses the "maturity" of an organization's software process and provides a qualitative indication of a maturity level.

Elements of a SPI Framework



Maturity Models

- A maturity model is applied within the context of an SPI framework.
- The intent of the maturity model is to provide an overall indication of the "process maturity" exhibited by a software organization

The SPI Process

Assessment and Gap Analysis

- *Assessment* examines a wide range of actions and tasks that will lead to a high quality process.
  - **Consistency.** Are important activities, actions and tasks applied consistently across all software projects and by all software teams?
  - **Sophistication.** Are management and technical actions performed with a level of sophistication that implies a thorough understanding of best practice?
  - **Acceptance.** Is the software process and software engineering practice widely accepted by management and technical staff?
  - **Commitment.** Has management committed the resources required to achieve consistency, sophistication and acceptance?
- *Gap analysis*—The difference between local application and best practice represents a "gap" that offers opportunities for improvement.

Education and Training

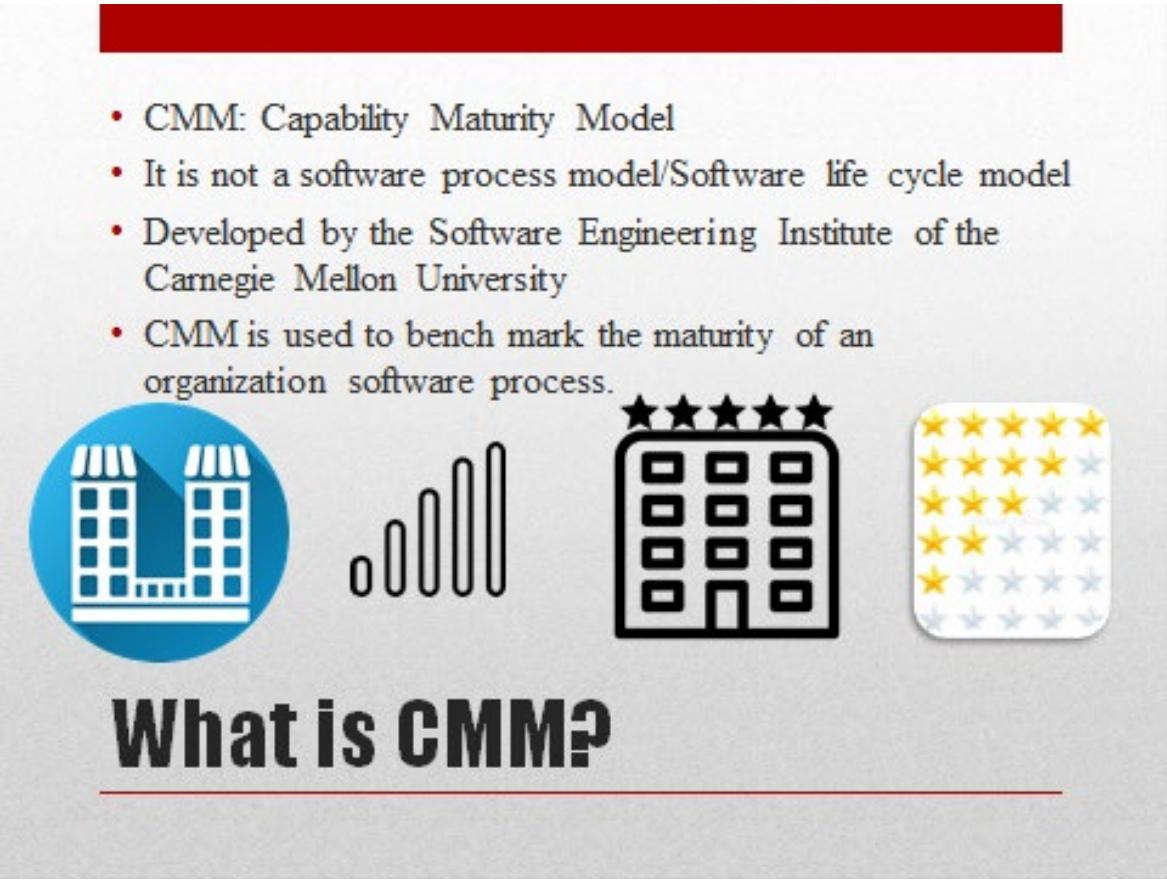
- Three types of education and training should be conducted:
  - **Generic concepts and methods.** Directed toward both managers and practitioners, this category stresses both process and practice. The intent is to provide professionals with the intellectual tools they need to apply the software process effectively and to make rational decisions about improvements to the process.
  - **Specific technology and tools.** Directed primarily toward practitioners, this category stresses technologies and tools that have been adopted for local use. For example, if UML has been chosen for analysis and design modeling, a training curriculum for software engineering using UML would be established.
  - **Business communication and quality-related topics.** Directed toward all stakeholders, this category focuses on "soft" topics that help enable better communication among stakeholders and foster a greater quality focus.

- **Selection and Justification**
  - choose the process model (Chapters 2 and 3) that best fits your organization, its stakeholders, and the software that you build
  - decide on the set of framework activities that will be applied, the major work products that will be produced and the quality assurance checkpoints that will enable your team to assess progress
  - develop a work breakdown for each framework activity (e.g., modeling), defining the task set that would be applied for a typical project
  - Once a choice is made, time and money must be expended to install it within an organization and these resource expenditures should be justified.

- **Installation/Migration**
  - actually *software process redesign* (SPR) activities. Scacchi [Sca00] states that "SPR is concerned with identification, application, and refinement of new ways to dramatically improve and transform software processes."
  - three different process models are considered:
    - the existing ("as-is") process,
    - a transitional ("here-to-there") process, and
    - the target ("to be") process.

- **Evaluation**
  - assesses the degree to which changes have been instantiated and adopted,
  - the degree to which such changes result in better software quality or other tangible process benefits, and
  - the overall status of the process and the organizational culture as SPI activities proceed
- From a qualitative point of view, past management and practitioner attitudes about the software process can be compared to attitudes polled after installation of process changes.

CMMI (CMM or The People CMM or P-CMM)



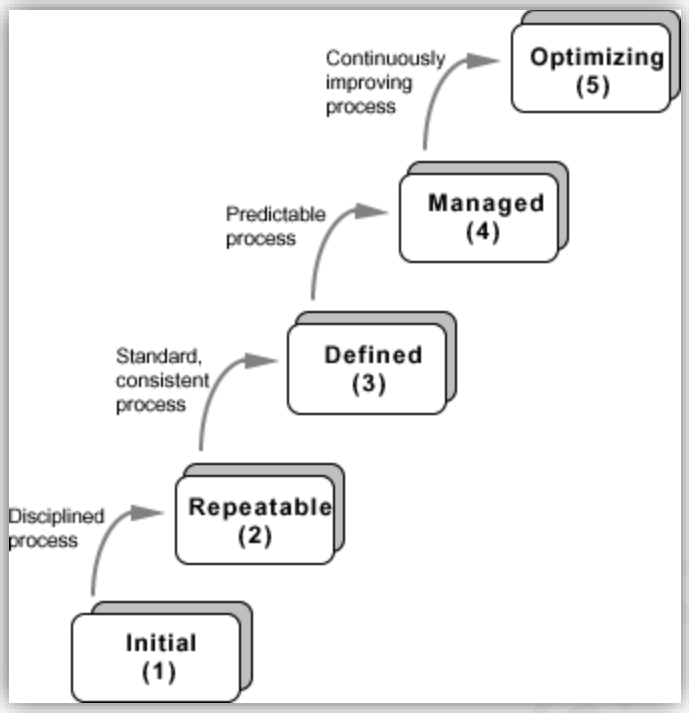
- Describes an evolutionary improvement path for software organizations from an ad hoc, immature process to a mature, disciplined one.
- Provides guidance on
    - how to gain control of processes for developing and maintaining software
    - how to evolve toward a culture of software engineering and management excellence.

What are the CMM Levels?

(The five levels of software process maturity)

Maturity level indicates level of process capability:

- Initial
- Repeatable
- Defined
- Managed
- Optimizing



Level 1: Initial

- Initial : The software process is characterized as ad hoc, and occasionally even chaotic.
- Few processes are defined, and success depends on individual effort.
    - At this level, frequently have difficulty making commitments that the staff can meet with an orderly process
    - Products developed are often over budget and schedule
    - Wide variations in cost, schedule, functionality and quality targets
    - Capability is a characteristic of the individuals, not of the organization

Level 2: Repeatable

- Basic process management processes are established to track cost, schedule, and functionality.
- The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
    - Realistic project commitments based on results observed on previous projects
    - Software project standards are defined and faithfully followed
    - Processes may differ between projects
    - Process is disciplined
    - earlier successes can be repeated

Level 3: Defined

- At this level the software process for both management and engineering activities are well defined and documented.

Level 4: Managed

- **Work is quantitatively controlled**
- Software quality management
    - Management can effectively controlled the software development effort using precise measurement
    - At this level set a quantitative quality goals for both software process and maintenance
- Quantitative process management
    - At this maturity level the performance of the processes is controlled using the statistical and other quantitative techniques and is quantitatively predictable

Level 5: Optimizing

- Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.
- Goal is to prevent the occurrence of defects
- Data on process effectiveness used for cost benefit analysis of new technologies and proposed process changes

ISO 9000

- International standard organization is an association of 60 countries
- It published a 9000 series standards in 1987
- ISO 9000 STD specifies the guidelines for maintaining the quality of the system
- It address the operational aspects and organizational aspects
    - Responsibilities
    - Reporting

- It specifies a set of recommendations for quality product development
- Three standards
    - ISO 9001
    - ISO 9002
    - ISO 9003
- These series of standards are based on the premise that,
    - If the process is followed for production, then good quality products are bounded automatically

ISO 9001

- This std is applied for
    - Design
    - Development
    - Production
    - Servicing of goods
- Applicable for S/W development

ISO 9002

- This std is applicable for organization which do not design products but only involved in production

- Eg. Car Manufacturing industries

Not applicable for S/W organizations

ISO 9003

- Applied to organizations involved in installation and testing of products

Cloud-based software

The cloud

- The cloud is made up of very large number of remote servers that are offered for rent by companies that own these servers.
  - Cloud-based servers are ‘virtual servers’, which means that they are implemented in software rather than hardware.
- You can rent as many servers as you need, run your software on these servers and make them available to your customers.
  - Your customers can access these servers from their own computers or other networked devices such as a tablet or a TV.
  - Cloud servers can be started up and shut down as demand changes.
- You may rent a server and install your own software, or you may pay for access to software products that are available on the cloud.

Table 5.1 Benefits of using the cloud for software development

Cost
You avoid the initial capital costs of hardware procurement

Startup time
You don't have to wait for hardware to be delivered before you can start work. Using the cloud, you can have servers up and running in a few minutes.

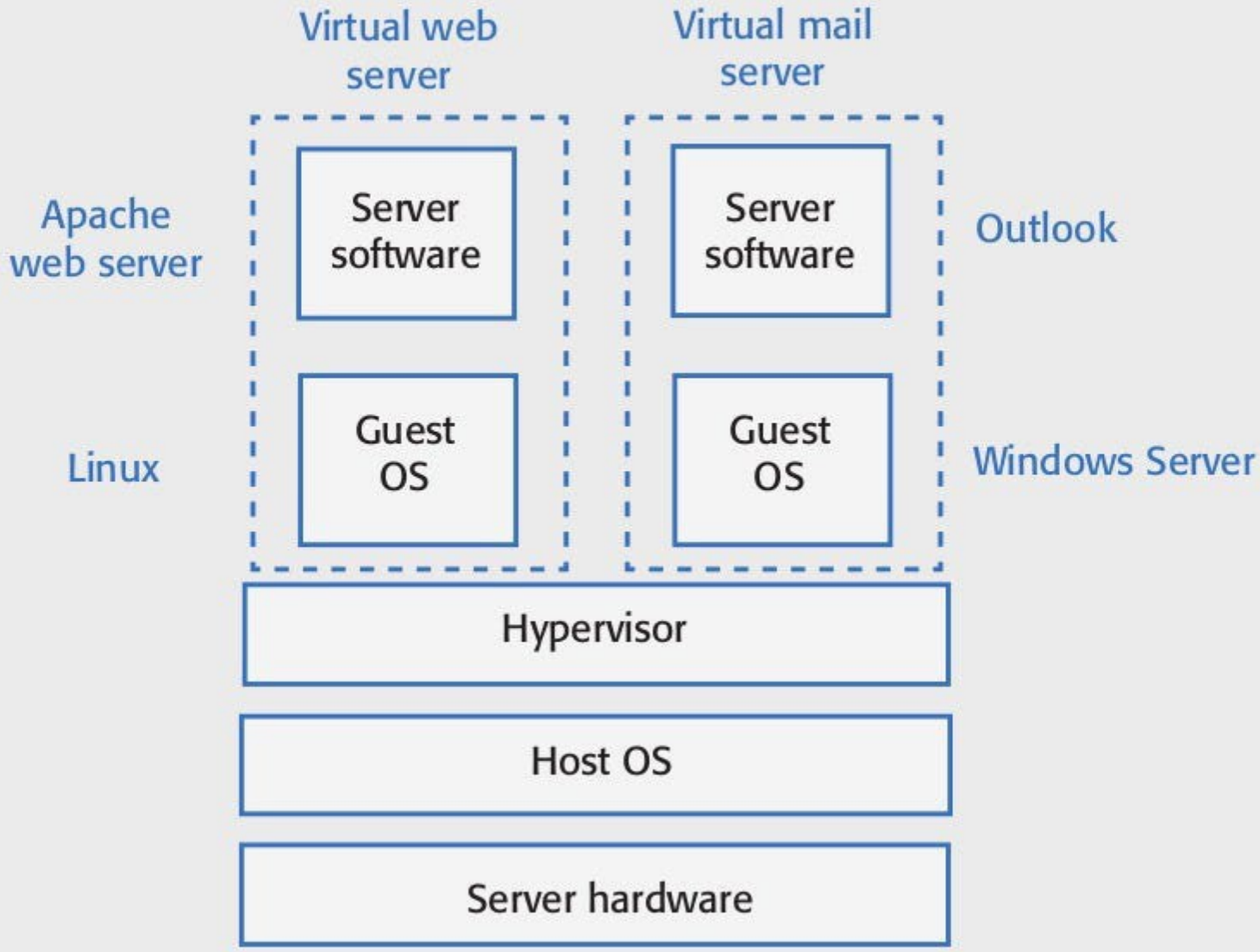
Server choice
If you find that the servers you are renting are not powerful enough, you can upgrade to more powerful systems. You can add servers for short-term requirements, such as load testing.

Distributed development
If you have a distributed development team, working from different locations, all team members have the same development environment and can seamlessly share all information.

Virtual cloud servers

- A virtual server runs on an underlying physical computer and is made up of an operating system plus a set of software packages that provide the server functionality required.
- A virtual server is a stand-alone system that can run on any hardware in the cloud.
  - This 'run anywhere' characteristic is possible because the virtual server has no external dependencies.
- Virtual machines (VMs), running on physical server hardware, can be used to implement virtual servers.
  - A hypervisor provides hardware emulation that simulates the operation of the underlying hardware.
- If you use a virtual machine to implement virtual servers, you have exactly the same hardware platform as a physical server.

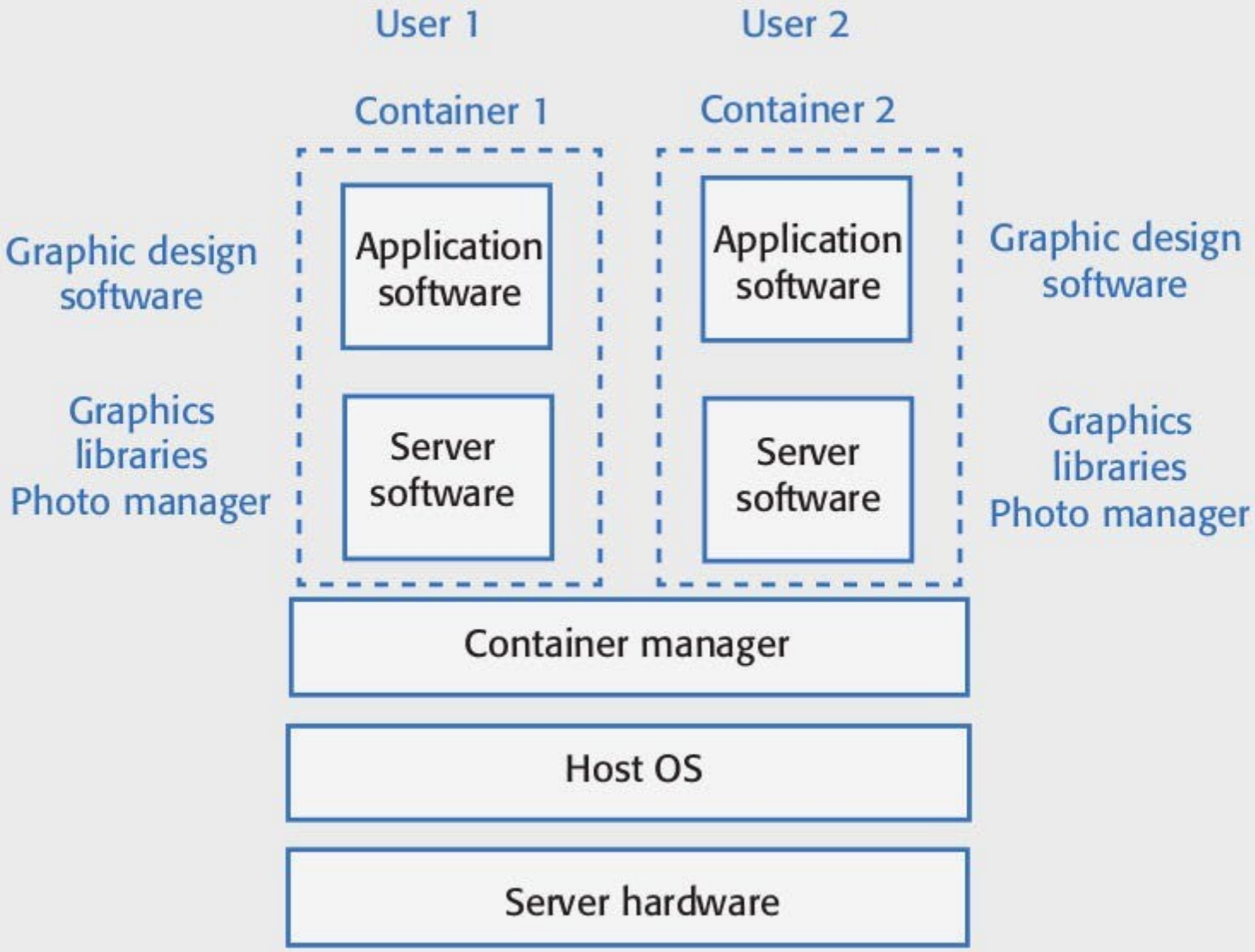
Figure 5.2 Implementing a virtual server as a virtual machine



Container-based virtualization

- If you are running a cloud-based system with many instances of applications or services, these all use the same operating system, you can use a simpler virtualization technology called ‘containers’.
- Using containers accelerates the process of deploying virtual servers on the cloud.
  - Containers are usually megabytes in size whereas VMs are gigabytes.
  - Containers can be started and shut down in a few seconds rather than the few minutes required for a VM.
- Containers are an operating system virtualization technology that allows independent servers to share a single operating system.
  - They are particularly useful for providing isolated application services where each user sees their own version of an application.

Figure 5.3 Using containers to provide isolated services



Docker

- Containers were developed by Google around 2007 but containers became a mainstream technology around 2015.
- An open-source project called Docker provided a standard means of container management that is fast and easy to use.
- Docker is a container management system that allows users to define the software to be included in a container as a Docker image.
- It also includes a run-time system that can create and manage containers using these Docker images.

Figure 5.4 The Docker container system

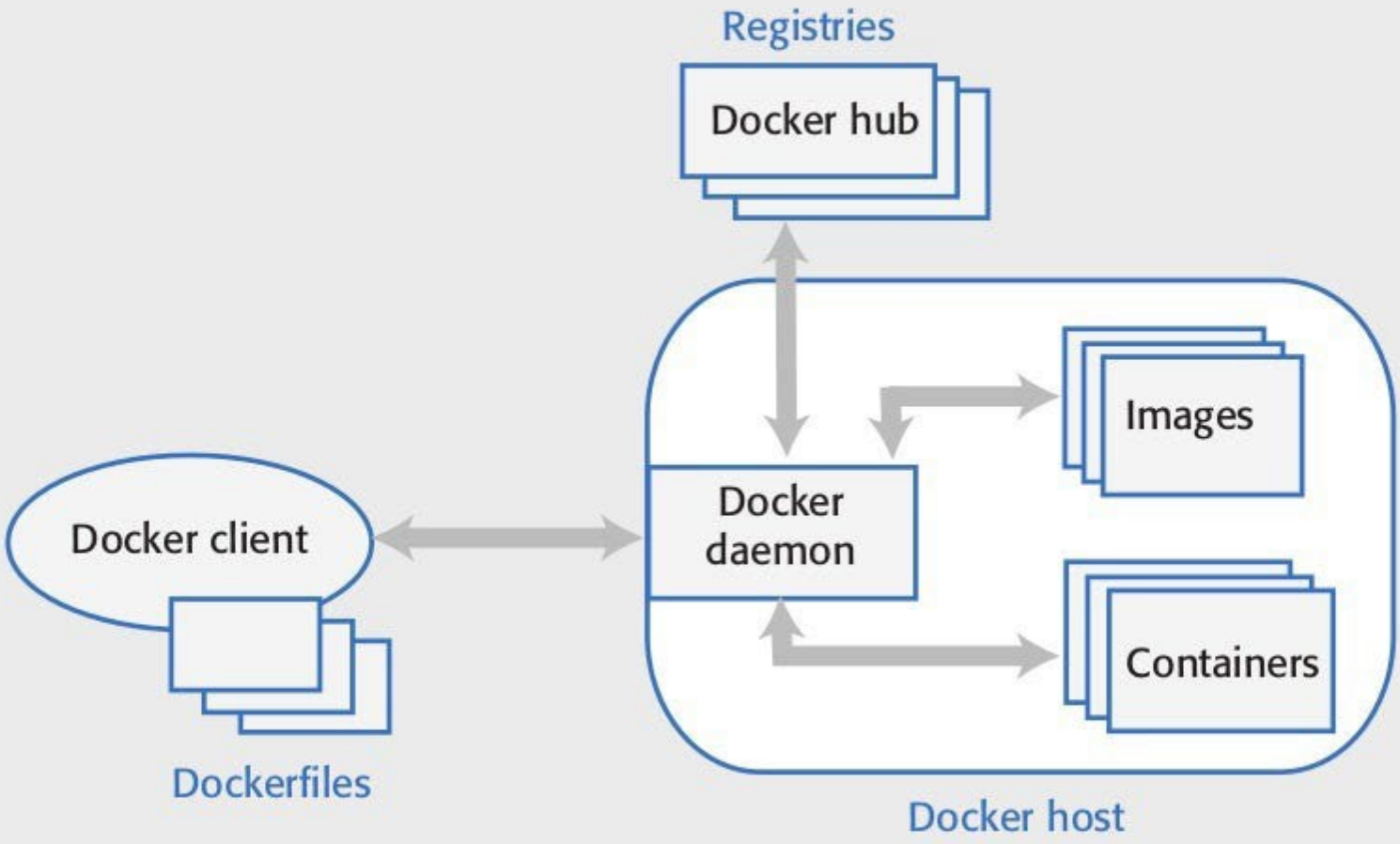


Table 5.2 The elements of the Docker container system

Docker daemon
This is a process that runs on a host server and is used to setup, start, stop, and monitor containers, as well as building and managing local images.

Docker client
This software is used by developers and system managers to define and control containers

Dockerfiles
Dockerfiles define runnable applications (images) as a series of setup commands that specify the software to be included in a container. Each container must be defined by an associated Dockerfile.

Image
A Dockerfile is interpreted to create a Docker image, which is a set of directories with the specified software and data installed in the right places. Images are set up to be runnable Docker applications.

Table 5.2 The elements of the Docker container system

Docker hub
This is a registry of images that has been created. These may be reused to setup containers or as a starting point for defining new images.

Containers
Containers are executing images. An image is loaded into a container and the application defined bby the image starts execution. Containers may be moved from server to server without modification and replicated across many servers. You can make changes to a Docker container (e.g. by modifying files) but you then must commit these changes to create a new image and restart the container.

Docker images

- Docker images are directories that can be archived, shared and run on different Docker hosts. Everything that's needed to run a software system - binaries, libraries, system tools, etc. is included in the directory.
- A Docker image is a base layer, usually taken from the Docker registry, with your own software and data added as a layer on top of this.
  - The layered model means that updating Docker applications is fast and efficient. Each update to the filesystem is a layer on top of the existing system.
  - To change an application, all you have to do is to ship the changes that you have made to its image, often just a small number of files.

Benefits of containers

- They solve the problem of software dependencies. You don't have to worry about the libraries and other software on the application server being different from those on your development server.
  - Instead of shipping your product as stand-alone software, you can ship a container that includes all of the support software that your product needs.
- They provide a mechanism for software portability across different clouds. Docker containers can run on any system or cloud provider where the Docker daemon is available.
- They provide an efficient mechanism for implementing software services and so support the development of service-oriented architectures.
- They simplify the adoption of DevOps. This is an approach to software support where the same team are responsible for both developing and supporting operational software.

Everything as a service

- The idea of a service that is rented rather than owned is fundamental to cloud computing.
- Infrastructure as a service (IaaS)
  - Cloud providers offer different kinds of infrastructure service such as a compute service, a network service and a storage service that you can use to implement virtual servers.
- Platform as a service (PaaS)
  - This is an intermediate level where you use libraries and frameworks provided by the cloud provider to implement your software. These provide access to a range of functions, including SQL and NoSQL databases.
- Software as a service (SaaS)
  - Your software product runs on the cloud and is accessed by users through a web browser or mobile app.

Figure 5.5 Everything as a service

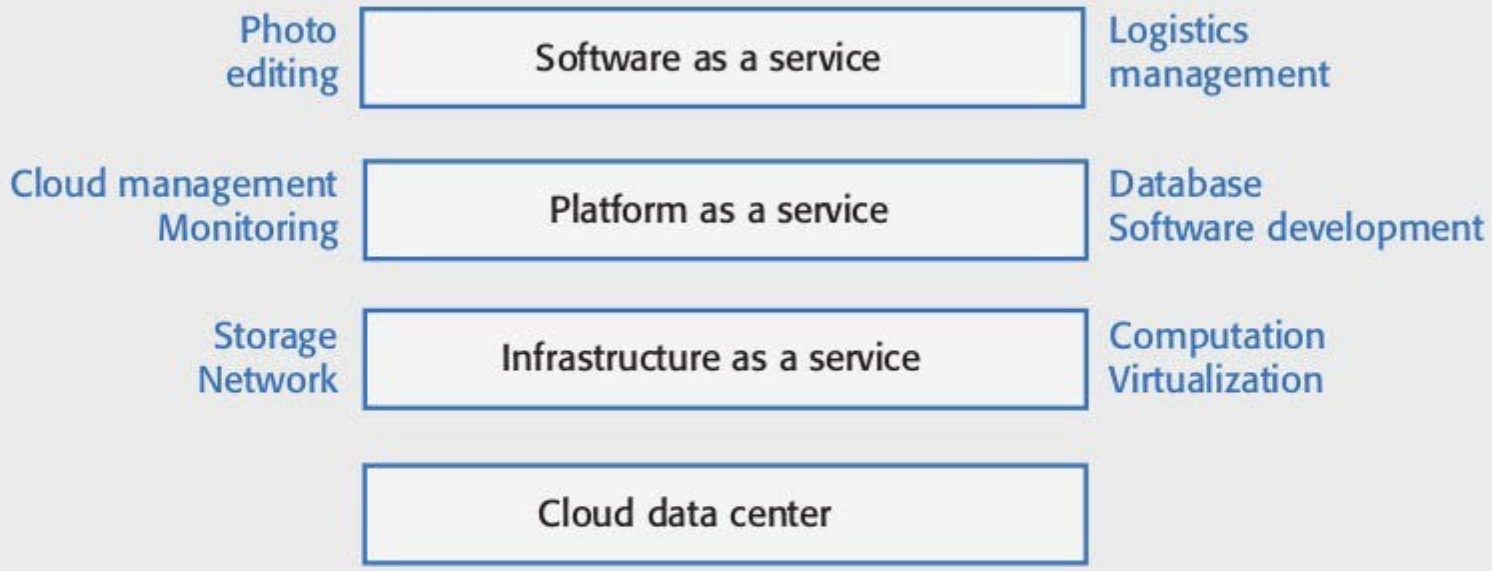
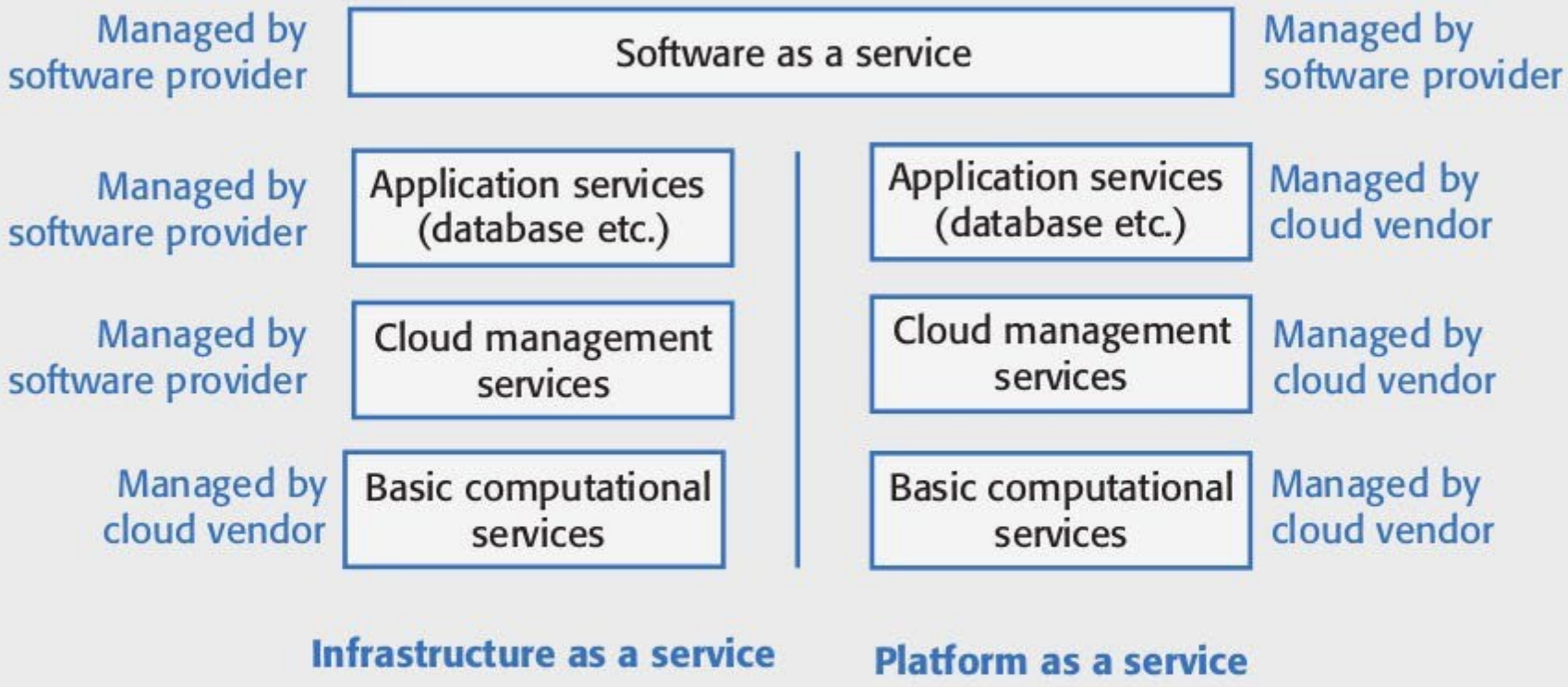


Figure 5.6 Management responsibilities for IaaS and PaaS



Software as a service

- Increasingly, software products are being delivered as a service, rather than installed on the buyer's computers.
- If you deliver your software product as a service, you run the software on your servers, which you may rent from a cloud provider.
- Customers don't have to install software and they access the remote system through a web browser or dedicated mobile app.
- The payment model for software as a service is usually a subscription model.
  - Users pay a monthly fee to use the software rather than buy it outright.

Figure 5.7 Software as a service

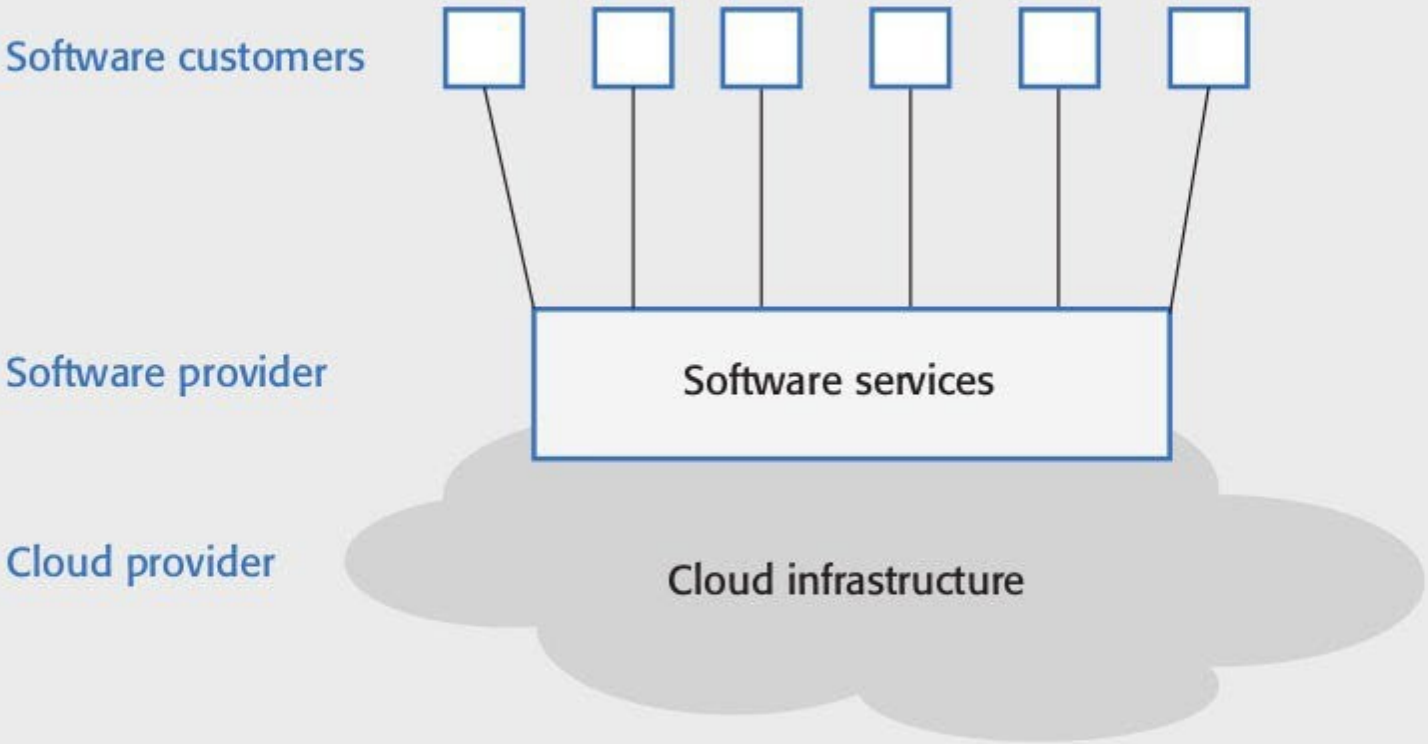


Table 5.3 Benefits of SaaS for software product providers

Cash flow
Customers either pay a regular subscription or pay as they use the software. This means you have a regular cash flow, with payments throughout the year. You don't have a situation where you have a large cash injection when products are purchased but very little income between product releases.

Update management
You are in control of updates to your product and all customers receive the update at the same time. You avoid the issue of several versions being simultaneously used and maintained. This reduces your costs and makes it easier to maintain a consistent software code base.

Continuous deployment
You can deploy new versions of your software as soon as changes have been made and tested. This means you can fix bugs quickly so that your software reliability can continuously improve.

Table 5.3 Benefits of SaaS for software product providers

Payment flexibility
You can have several different payment options so that you can attract a wider range of customers. Small companies or individuals need not be discouraged by having to pay large upfront software costs.

Try before you buy
You can make early free or low-cost versions of the software available quickly with the aim of getting customer feedback on bugs and how the product could be approved.

Data collection
You can easily collect data on how the product is used and so identify areas for improvement. You may also be able to collect customer data that allows you to market other products to these customers.

Figure 5.8 Advantages and disadvantages of SaaS for customers