MODULE 5

Software Quality, Process Improvement and Technology trends

Software Quality, Software Quality Dilemma, Achieving Software Quality Elements of Software Quality Assurance, SQA Tasks ,Software measurement and metrics. Software Process Improvement(SPI), SPI Process CMMI process improvement framework, ISO 9001:2000 for Software. Cloud-based Software - Virtualisation and containers, Everything as a service(IaaS, PaaS), Software as a service. Microservices Architecture - Microservices, Microservices architecture, Microservice deployment.

SOFTWARE QUALITY

Two kinds of quality are sought out

- **Quality of design**
    - The characteristic that designers specify for an item
    - This encompasses requirements, specifications, and the design of the system
- **Quality of conformance (i.e., implementation)**
    - The degree to which the design specifications are followed during manufacturing
    - This focuses on how well the implementation follows the design and how well the resulting system meets its requirements.
    - Quality also can be looked at in terms of user satisfaction User satisfaction=compliant product+good quality + delivery within budget and schedule.
- **Kinds of Quality Costs**
    - **Prevention costs**
        - Quality planning, formal technical reviews, test equipment, training
    - **Appraisal costs**
        - Inspections, equipment calibration and maintenance, testing
    - **Failure costs** – subdivided into internal failure costs and external failure costs

- **Internal failure costs**
  - Incurred when an error is detected in a product prior to shipment
  - Include rework, repair, and failure mode analysis
- **External failure costs**
  - Involves defects found after the product has been shipped
- Include complaint resolution, product return and replacement, help line support, and warranty work

- **SOFTWARE QUALITY DEFINITION**
  - 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.
  - **Effective process**: establishes the infrastructure that supports any effort that building a high quality software product .The management aspects of process create the checks and balances that help avoid project chaos—a key contributor to poor quality.
  - **Useful product**:

    delivers the content, functions, and features that the end-user desires in a reliable, error free way.

    Satisfies requirements of stakeholders and that are expected of all high quality software.
  - **Adding value:**
    - high quality software provides benefits to producer and user
    - To producer: less maintenance effort, fewer bug fixes, and reduced customer support
    - To user: expedites some business process.
    - End results: Increased revenue, better profitability when an application supports a business process, and/or improved availability of information that is crucial for the business.

- **QUALITY DIMENSIONS AND FACTORS**

  Can be used as generic quality indicators of a software product.
  - ***Garvin Quality Dimensions***
  - ***McCall's Quality Factors***
  - ***ISO 9126 Quality Factors***
  - ***Targeted Factors***

Garvin	McCall	ISO 9126	Targeted
Performance Quality **Feature Quality** **Reliability** **Conformance** **Durability** **Serviceability** **Aesthetic** **Perception**	**Correctness** **Reliability** **Efficiency** **Integrity** **Usability** **Maintainability** **Flexibility** **Testability** **Portability** **Reusability** **Interoperability**	**Functionality** **Reliability** **Usability** **Efficiency** **Maintainability** **Portability**	1. **Intuitiveness** 2. **Efficiency** 3. **Robustness** 4. **Richness**

- **David Garvin [Gar87]:**
  - Performance Quality. Does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end-user?
  - Feature quality. Does the software provide features that surprise and delight first-time end-users?
  - Reliability. Does the software deliver all features and capability without failure? Is it available when it is needed? Does it deliver functionality that is error free?
  - Conformance. Does the software conform to local and external software standards that are relevant to the application? Does it conform to de facto

design and coding conventions? For example, does the user interface conform to accepted design rules for menu selection or data input?

- Durability. Can the software be maintained (changed) or corrected (debugged) without the inadvertent generation of unintended side effects? Will changes cause the error rate or reliability to degrade with time?
- Serviceability. Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period. Can support staff acquire all information they need to make changes or correct defects?
- Aesthetics. Most of us would agree that an aesthetic entity has a certain elegance, a unique flow, and an obvious "presence" that are hard to quantify but evident nonetheless.
- Perception. In some situations, you have a set of prejudices that will influence your perception of quality.

Other quality dimensions

- Efficiency
  - The degree to which the software makes optimal use of software resources.
- Usability
  - The degree to which the software is easy to learn, use, operate, prepare input for and interpret output from.
- Maintainability
  - The ease with which repair maybe made to the software
- Reliability
  - The amount of time that the software is available for use

The Software Quality Dilemmas

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway.
- 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.

Achieving Software Quality

- **Broad activities that help a software team achieve high quality software:**

    1. ***Quality assurance (QA)*** *– establishes the infrastructure that supports solid software engineering methods, rational project management, and quality control actions.*

    2. ***Quality control (QC)*** *– action that helps ensure each work products meets its quality goals (e.g., Review design models to ensure that they are complete and consistent).*

    3. ***Software engineering method*** *– understand the problem to be solved, create a design that conforms to the problems and exhibit characteristics that lead to software that are reliable, efficient, usable, etc.*

    4. ***Project management technique*** *–use estimation to verify that delivery dates are achievable, schedule dependencies are understood and conduct risk planning so that problem do not breed chaos.*

Software Quality Assurance (SQA)

- Encompasses:

    1. An SQA process

    2. Specific QA and QC tasks – technical review, audits, multitier testing strategy etc.

    3. Effective SE practice (methods and tools) – risk management

    4. Control of all software work products and changes made to them – change management, security management

    5. A procedure to ensure compliance with standards – IEEE, ISO, CMMI, Six Sigma etc

    6. Measurement and reporting mechanisms – SQA group

Elements of Software Quality Assurance

Standards.

The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.

Reviews and audits.

Technical reviews are a quality control activity per-formed by software engineers for software engineers. Their intent is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work.

Testing.

Software testing (Chapters 22 through 26) is a quality control function that has one primary goal—to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal. Error/defect collection and analysis. The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.
Change management.

Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality. SQA ensures that adequate change management practices have been instituted.

Education.

Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement and is akey proponent and sponsor of educational programs.

Vendor management.

Three categories of software are acquired from external software vendors—shrink-wrapped packages (e.g., Microsoft Office), a tailored shell that provides a basic skeletal structure that is custom tailored to the needs of a purchaser, and contracted software that

is custom designed and constructed from specifications provided by the customer organization.

Security management.

With the increase in cyber crime and new government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection for WebApps, and ensure that software has not been tampered with internally. SQA ensures that appropriate process and technology are used to achieve software security.

Safety.

Because software is almost always a pivotal component of human-rated systems (e.g., automotive or aircraft applications), the impact of hidden defects can be catastrophic. SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.

Risk management.

Although the analysis and mitigation of risk is the concern of software engineers, the SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

SQA TASKS

1. Prepares an SQA plan for a project.
   - The plan identifies:
     - evaluations to be performed
     - audits and reviews to be performed
     - standards that are applicable to the project
     - procedures for error reporting and tracking
     - documents to be produced by the SQA group
     - amount of feedback provided to the software project team
2. Participates in the development of the project's software process description.

The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.

3. Reviews software engineering activities to verify compliance with the defined software process.
    - identifies, documents, and tracks deviations from the process and verifies that corrections have been made.
4. Audits designated software work products to verify compliance with those defined as part of the software process.
    - reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made periodically reports the results of its work to the project manager.
5. Ensures that deviations in software work and work products are documented and handled according to a documented procedure.
6. Records any noncompliance and reports to senior management.
    - Noncompliance items are tracked until they are resolved.

SQA GOALS

- **Requirements quality**. The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow.
- **Design quality**. Every element of the design model should be assessed by the software team to ensure that it exhibits high quality and that the design itself conforms to requirements.
- **Code quality**. Source code and related work products (e.g., other descriptive information) must conform to local coding standards and exhibit characteristics that will facilitate maintainability.
- **Quality control effectiveness**. A software team should apply limited resources in a way that has the highest likelihood of achieving a high quality result.

SOFTWARE MEASUREMENT AND METRICS

Measurement—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs; can be used in conjunction with all other framework and umbrella activities.

Direct measures of the software process include cost and effort applied. Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time.

Indirect measures of the product include functionality, quality, complexity, efficiency, reliability, maintainability,

Software metrics is a standard of measure that contains many activities which involve some degree of measurement. It can be classified into three categories: product metrics, process metrics, and project metrics.

- **Product metrics** describe the characteristics of the product such as size, complexity, design features, performance, and quality level.
- **Process metrics** can be used to improve software development and maintenance. Examples include the effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process.
- **Project metrics** describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.
- **Size-oriented software metrics** are derived by normalizing quality and/or productivity measures by considering the size of the software that has been produced. Eg: lines of code
- **Function-oriented software metrics** use a measure of the functionality delivered by the application as a normalization value. The most widely used function-oriented metric is the function point (FP).

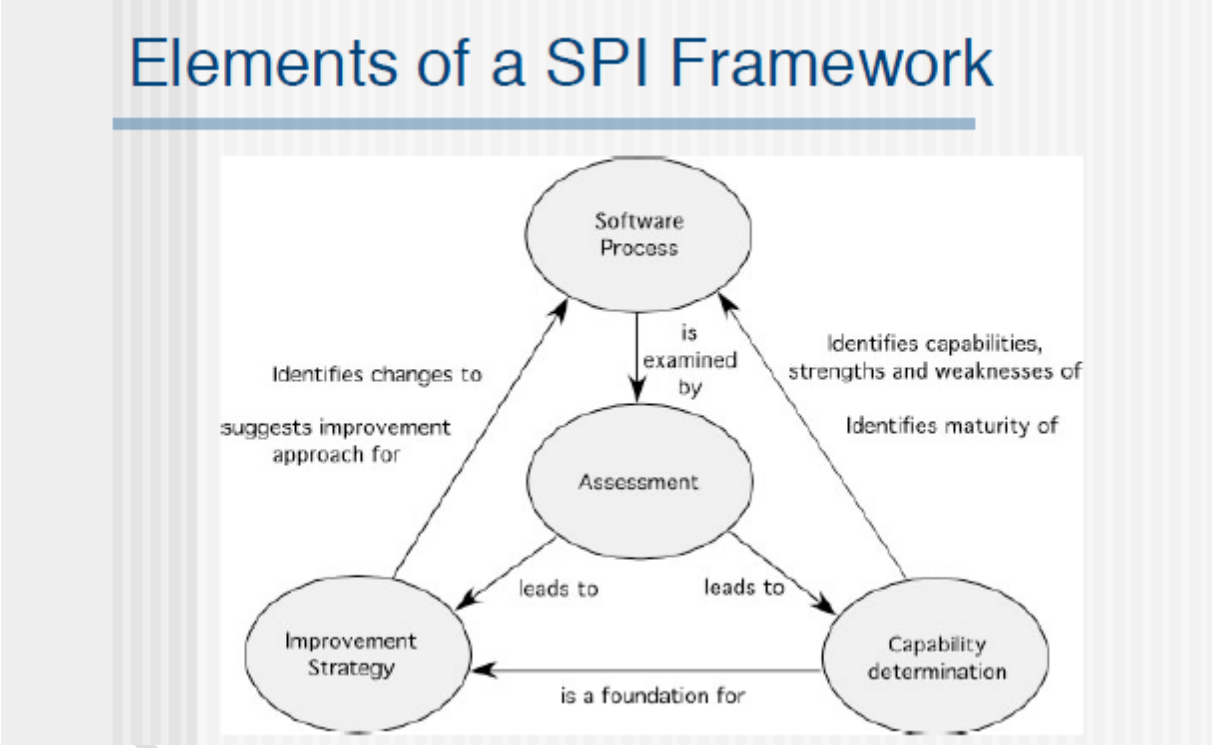
- **Webapp metrics** : No of static pages,number of dynamic pages.
- Measurement enables managers and practitioners to improve the software process; assist in the planning, tracking, and control of software projects; and assess the quality of the product (software) that is produced. Measures of specific attributes of the process, project, and product are used to compute software metrics. These metrics can be analyzed to provide indicators that guide management and technical actions.
  Process metrics enable an organization to take a strategic view by providing insight into the effectiveness of a software process. Project metrics are tactical. They enable a project manager to adapt project workflow and technical approach in a real-time manner.
  Both size- and function-oriented metrics are used throughout the industry. Size-oriented metrics use the line of code as a normalizing factor for other measures such as person-months or defects. The function point is derived from measures of the information domain and a subjective assessment of problem complexity. In addition, object-oriented metrics and Web application metrics can be used. Software quality metrics, like productivity metrics, focus on the process the project, and the product. By developing and analyzing a metrics baseline for

quality, an organization can correct those areas of the software process that are the cause of software defects.

SOFTWARE PROCESS IMPROVEMENT

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 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 resourceexpenditures should be justified.

Installation/Migration

 actually *software process redesign* (SPR) activities.Scacchi [Sca00] states that "SPR is concerned withidentification, 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 beeninstantiated and adopted,
 the degree to which such changes result in better softwarequality 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 becompared to attitudes polled after installation of process changes.

Risk Management

manage risk at three key points in the SPI process :
 prior to the initiation of the SPI roadmap,
 during the execution of SPI activities (assessment, education,selection, installation), and
 during the evaluation activity that follows the instantiation of some process characteristic.

CMMI MATURITY MODEL

The Capability Maturity Model Inegration (CMM) is **a methodology used to develop and refine an organization's software development process.**

LEVELS OF CMMI

- **Maturity Level 0 – Incomplete:** At this stage work "may or may not get completed." Goals have not been established at this point and processes are only partly formed or do not meet the organizational needs.

- **Maturity Level 1 – Initial:** Processes are viewed as unpredictable and reactive. At this stage, "work gets completed but it's often delayed and over budget." This is the worst stage a business can find itself in — an unpredictable environment that increases risk and inefficiency.

- **Maturity Level 2 – Managed:** There's a level of project management achieved. Projects are "planned, performed, measured and controlled" at this level, but there are still a lot of issues to address.

- **Maturity Level 3 – Defined:** At this stage, organizations are more proactive than reactive. There's a set of "organization-wide standards" to "provide guidance across projects, programs and portfolios." Businesses understand their shortcomings, how to address them and what the goal is for improvement.

- **Maturity Level 4 – Quantitatively managed:** This stage is more measured and controlled. The organization is working off quantitative data to determine predictable processes that align with stakeholder needs. The business is ahead of risks, with more data-driven insight into process deficiencies.

- **Maturity Level 5 – Optimizing:** Here, an organization's processes are stable and flexible. At this final stage, an organization will be in constant state of improving and responding to changes or other opportunities. The organization is stable, which allows for more "agility and innovation," in a predictable environment.

THE ISO 9000 QUALITY STANDARDS

The ISO 9000 is a family of standards primarily concerned with "quality management".

The ISO 9000series of standards certify the procedures/steps taken by an organization to concurrently fulfill the followings: (a)the customer's quality requirements, (b)meet the applicable regulatory requirements,(c)aiming to enhance customer satisfaction, and (d) achieve continual improvement of its performance in pursuit of these objectives.

ISO 9001:2000 for Software—a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies.

ISO 9001:2000 stresses the importance for an organization to identify, implement, manage and continually improve the effectiveness of the processes that are necessary for the quality management system, and to manage the interactions of these processes in order to achieve the organization's objectives . Process effectiveness and efficiency can be accessed through internal or external review processes and be evaluated on a maturity scale.

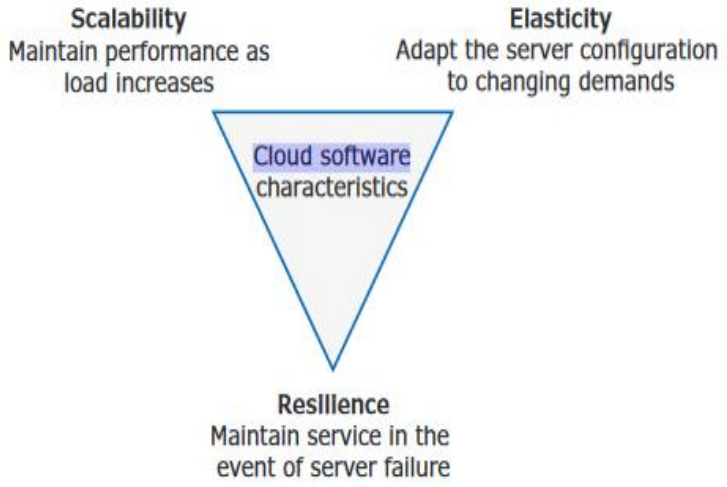
CLOUD BASED SOFTWARES

- The cloud is a very large number of remote servers that are offered for rent by companies that own these servers. 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. You may rent a server and install your own software, or you may pay for access to software products
- The remote servers are **"virtual servers**," which means they are implemented in software rather than hardware. Many virtual servers can run simultaneously on each cloud hardware node, using virtualization support that is built in to the hardware. Running multiple servers has very little effect on server performance. The hardware is so powerful that it can easily run several virtual servers at the same time.
- Cloud companies such **as Amazon and Google** provide cloud management software that makes it easy to acquire and release servers on demand. You

can automatically upgrade the servers that you are running, and the cloud management software provides resilience in the event of a server failure.

- **Scalability** reflects the ability of your software to cope with increasing numbers of users. As the load on your software increases, the software automatically adapts to maintain the system performance and response time.
- **Elasticity** is related to scalability but allows for scaling down as well as scaling up. That is, you can monitor the demand on your application and addor remove servers dynamically as the number of users changes. This means that you pay for only the servers you need, when you need them.
- **Resilienc**e means that you can design your software architecture to toler-ate server failures. You can make several copies of your software available concurrently. If one of these fails, the others continue to provide a service. You can cope with the failure of a cloud data center by locating redundant servers in different places

Figure 5.1 Scalability, elasticity, and resilience



BENEFITS OF USING CLOUD FOR SOFTWARE DEVLOPMENT

Table 5.1 Benefits of using the cloud for software development

Factor	Benefit
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.

Virtualization and containers

- All cloud servers are virtual servers.
- **A virtual server** runs on an underlying physical computer and is made up of an operating system plus a set of soft-ware packages that provide the server functionality required.
- The general idea is that a virtual server is a stand-alone system that can run on any hardware in the cloud.
- When you run software on different computers, you often encounter problems because some of the external software that you rely on is unavailable or is different in some way from the version that you're using. If you use a virtual server, you avoid these problems. You load all of the software that you need, so you are not relying on software being made available by someone else.
- **Virtual machines (VMs),** running on physical server hardware, can be used to implement virtual servers (Figure 5.2). The details are complex, but you can think of the hypervisor as providing a hardware emulation that simulates the operation of the underlying hardware.
- **The advantage of using a virtual machine to implement virtual servers** is that you have exactly the same hardware platform as a physical server. You can therefore run different operating systems on virtual machines that are hosted on the same computer. For example, Figure 5.2 shows that Linux and Windows can run concurrently on separate VM

Figure 5.2 Implementing a virtual server as a virtual machine

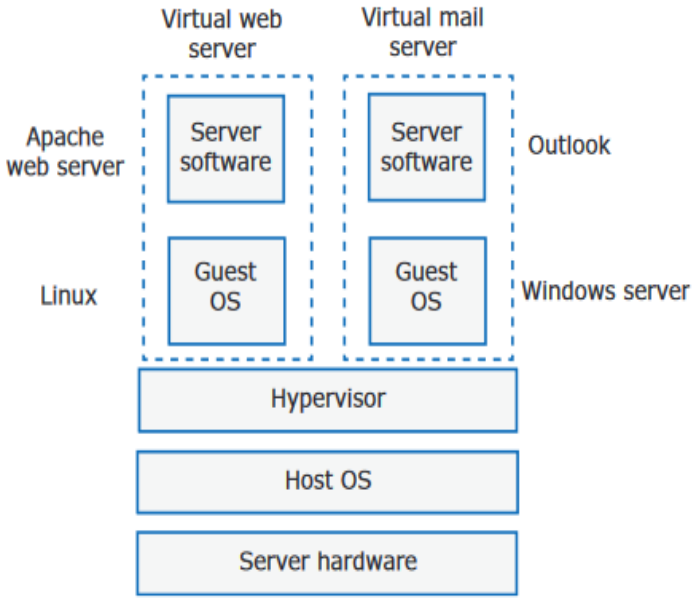
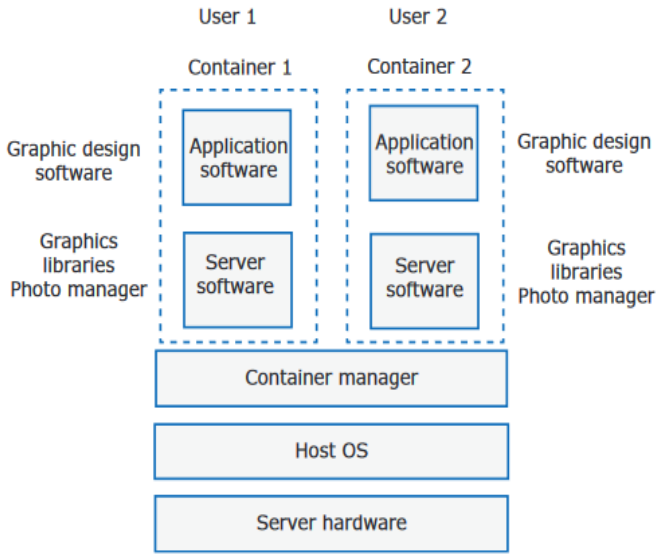


Figure 5.3 Using containers to provide isolated services



- The problem with implementing virtual servers on top of VMs is that creating a VM involves loading and starting up a large and complex operating system (OS).
- The time needed to install the OS and set up the other software on the VM is typically between 2 and 5 minutes on public cloud providers such as AWS.
- If you are running a cloud-based system with many instances of applications or services, these all use the same operating system. To cater to this situation, asimpler, lightweight, virtualization technology called **"containers"** may be used.

Benefits of containers

- Using containers dramatically speeds up the process of deploying virtual servers on the cloud.
- Containers are usually megabytes in size, whereas VMsare gigabytes. Containers can be started up and shut down in a few seconds rather than the few minutes required for a VM.
- Many companies that provide cloud-based software have now switched from VMs to containers because containers are faster to load and less demanding of machine resources.
-

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.
Example : Docker .

VM Vs Containers

Containers are a lightweight mechanism for running applications in the cloud and are particularly effective for running small applications such as stand-alone services. If your application depends on a large, shared database that provides continuous service, running this database on a VM is still the best option.

VMs and containers can coexist on the same physical system, so applications running in containers can access the database efficiently.

CLOUD SERVICES

Infrastructure as a service (IaaS) :This is a basic service level that all major cloud providers offer. They provide different kinds of infrastructure service, such as a compute service, a network service, and a storage service.

These infrastructure services may be used to implement virtual cloud-based servers.

The key benefits of using IaaS are that you don't incur the capital costs of buying hardware and you can easily migrate your software from one server to a more powerful server.

You are responsible for installing the software on the server, although many preconfigured packages are available to help with this. Using the cloud provider's control panel, you can easily add more servers if you need to as the load on your system increases.

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, includ-ing SQL and NoSQL databases. Using PaaS makes it easy to developauto-scaling software. You can implement your product so that as theload increases, additional compute and storage resources are addedautomatically.

Software as a service (SaaS) Your software product runs on the cloud andis accessed by users through a web browser or mobile app. We all knowand use this type of cloud service—mail services such as Gmail, storageservices such as Dropbox, social media services such as Twitter, and soon.

An important difference between IaaS and PaaS is the allocation of system management responsibilities.

Figure 5.7 Software as a service

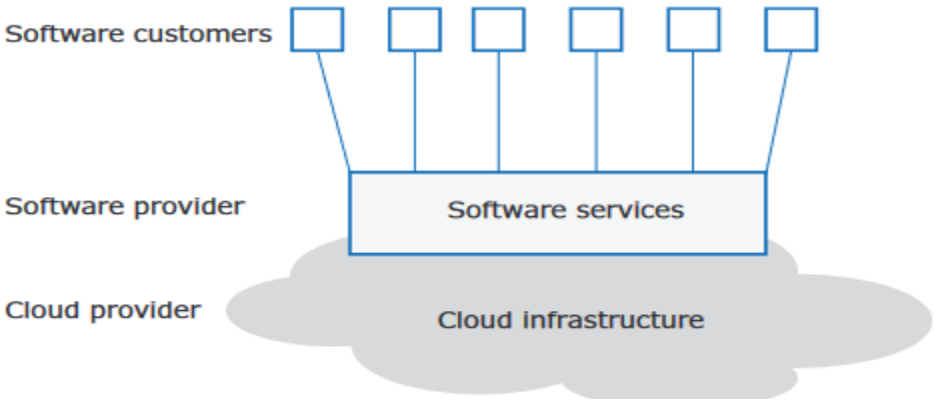
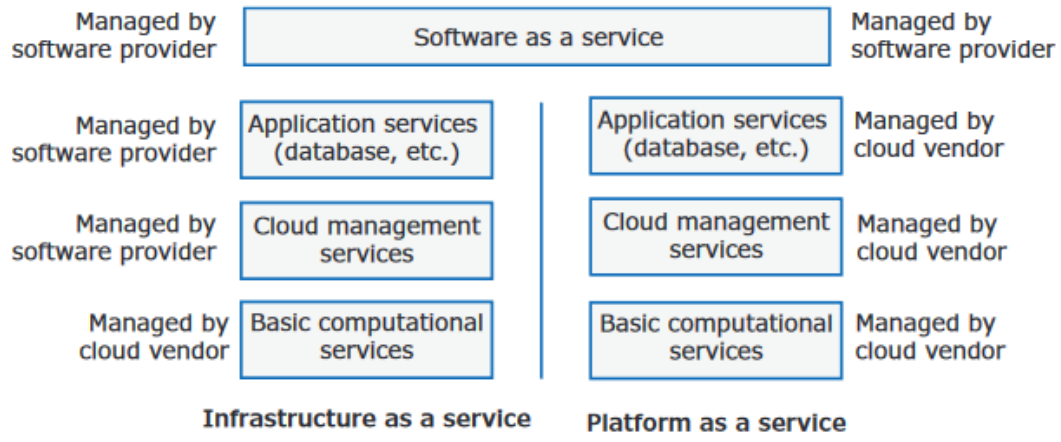


Figure 5.6 Management responsibilities for SaaS, IaaS, and PaaS



Benefits of SaaS

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.
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 allow you to market other products to these customers.

MICROSERVICES

A software service is a software component that can be accessed from remote computers over the Internet.

Given an input, a service produces a corresponding output, without side effects. The service is accessed through its published interface and all details of the service implementation are hid-den.

The manager of a service is called the **service provider,** and the user of a service is called a **service requestor**.

Microservices are small-scale services that may be combined to create applications. They should be independent, so that the service interface is not affected by changes to other services. It should be possible to modify the service and re-deploy it without changing or stopping other services in the system.

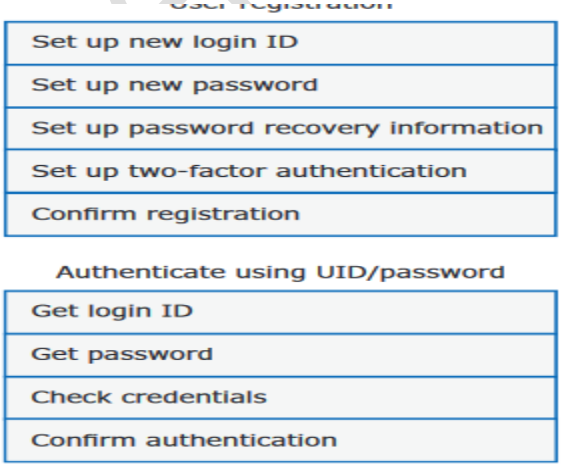
Microservices in a system can be implemented using different programming languages an database technologies.

A microservices architecture is based on services that are fine-grain components with a single responsibility.

For example, a coarse-grain authentication component or service might manage user names, check passwords, handle forgotten passwords, and send texts for two-factor authentication. In a microservice-based system, you may have individual microservices for each of these, such as get-login-name, check-password, and so on.

Consider a system that uses an authentication module that provides the following features:

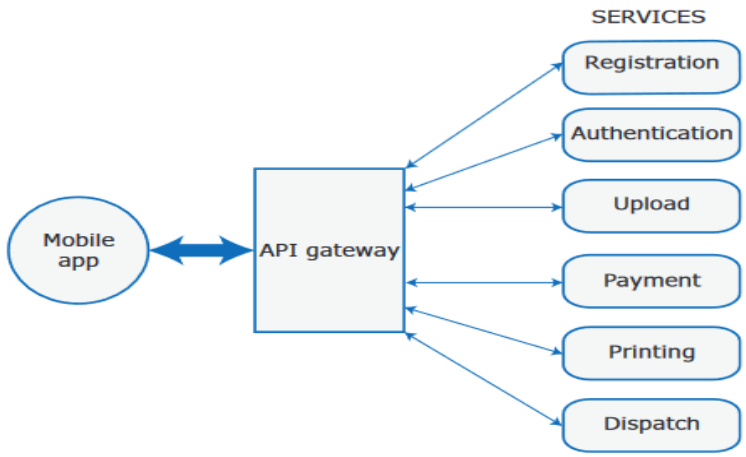
- user registration, where users provide information about their identity,security information, mobile (cell) phone number, and email address;
- authentication using user ID (UID)/password;
- two-factor authentication using code sent to mobile phone;
- user information management—for example, ability to change password or mobile phone number;
- password reset.



Microservices architecture

Imagine that you are developing a photo-printing service for mobile devices. Users can upload photos to your server from their phone or specify photos from their Instagram account that they would like to be printed. Prints can be made at different sizes and on different media.
Users can choose print size and print medium. For example, they may decide to print a picture onto a mug or a T-shirt. The prints or other media are prepared and then posted to their home. They pay for prints either using a payment service such as Android or Apple Pay or by registering a credit card with the printing service provider.

Figure 6.5 A microservices architecture for a photo-printing system



1. When a monolithic architecture is used, the whole system has to be rebuilt, retested, and re-deployed when any change is made. This can be a slow process, as changes to one part of the system can adversely affect other components. Frequent application updates are therefore impossible.

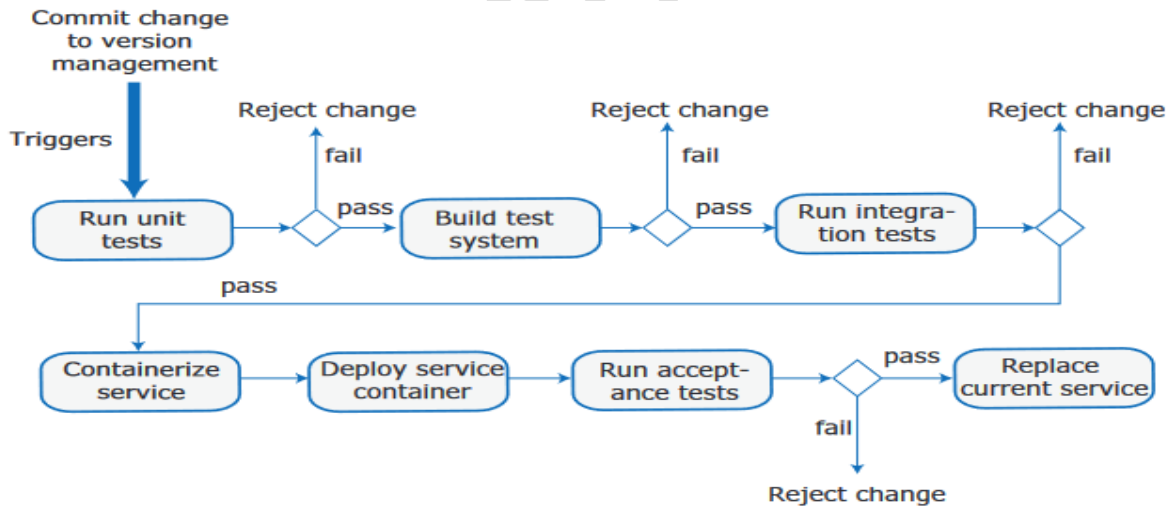
2. As the demand on the system increases, the whole system has to be scaled, even if the demand is localized to a small number of system components that implement the most popular system functions. Larger servers must be used, and this significantly increases the costs of renting cloud servers to run the software. Depending on how virtualization is managed, starting up a larger server can take several minutes, with the system service degraded until the new server is up and running.

Table 6.1 Characteristics of microservices

Characteristic	Explanation
Self-contained	Microservices do not have external dependencies. They manage their own data and implement their own user interface.
Lightweight	Microservices communicate using lightweight protocols, so that service communication overheads are low.
Implementation independent	Microservices may be implemented using different programming languages and may use different technologies (e.g., different types of database) in their implementation.
Independently deployable	Each microservice runs in its own process and is independently deployable, using automated systems.
Business-oriented	Microservices should implement business capabilities and needs, rather than simply provide a technical service.

MICROSERVICE DEPLOYMENT

when a microservices architecture is used, it is now normal practice for the development team to be responsible for deployment and ser-vice management as well as software development. This approach is known as DevOps—a combination of "development" and "operations.



A general principle of microservice-based development is that the ser-vice development team has full responsibility for their service, including the responsibility of deciding when to deploy new versions of that service. Good practice in this area is now to adopt a policy of **continuous deployment**. Con-tinuous deployment means that as soon as a change to a service has been made and validated, the modified service is re-deployed.

IMPORTANT QUESTIONS

1. Explain the elements of SQA and SQA tasks?
2. Illustrate SPI process.
3. List the levels of CMMI
4. How does software projects benefit from container deployment and microservice deployment?
5. Discuss the software quality dilemma ?
6. List and explain quality dimensions?
7. Compare Virtual machines and containers?
8. Differentiate IaaS,SaaS,PaaS with example ?
9. What is microservice?
10. Discuss the benefits of cloud for software development ?
11. Explain about microservice architecture?
12. Compare CMMI and ISO 9001:2000

ISO 9001 is an internationally recognized standard for quality management systems. While CMMI is a Carnegie Mellon University registered trade mark.

ISO 9001 has specific requirements for documented procedures for the control of documents, control of records, control of nonconforming products, internal audits, corrective actions and preventive actions. In addition, a quality policy, measurable objectives, and management reviews are required.

CMMI is focused on process improvement, while ISO 9001 focuses on customer satisfaction, process improvement, product conformity and the continual improvement of the quality management system. An organization could be CMMI certified or "capable" as mentioned in the inquiry, but still be some distance way from readiness for ISO 9001 certification.