

# MODULE 5

# Syllabus

- S/w Quality, Dilemma, **elements, Quality assurance, SQA** task, **SPI, CMMI** Frame work, ISO 9001:2000 standards, Cloud based S/w, Microservice architecture

# SOFTWARE QUALITY

- Software quality product is defined in term of its **fitness of purpose**. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document.

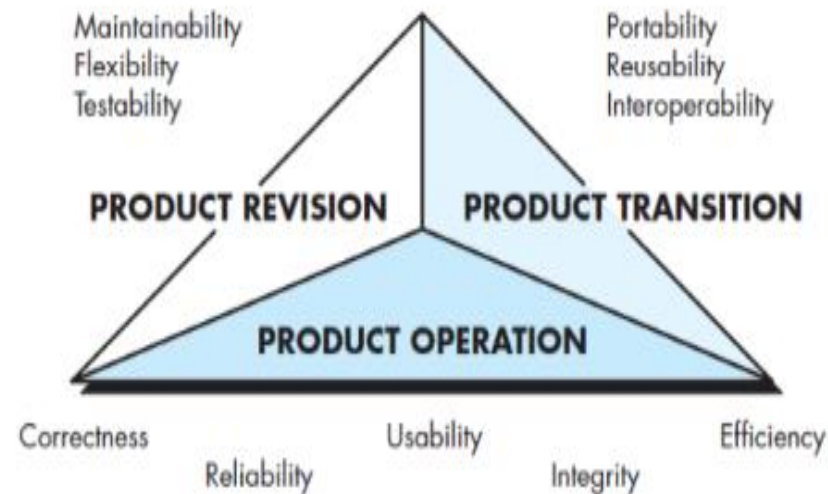
# Garvin's Quality Dimensions

- David Garvin suggests that quality should be considered by taking a multidimensional viewpoint
- ➤ **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?
- ➤ **Serviceability:** Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period?

# McCall's Quality Factors

- McCall, Richards, and Walters propose a useful categorization of factors that affect software quality. These software quality factors focus on three important aspects.



1. **Reliability.** The extent to which a program can be expected to perform its intended function with required precision.
2. **Efficiency:** The amount of computing resources and code required by a program to perform its function.
3. **Integrity.** Extent to which access to software or data by unauthorized persons can be controlled.
4. **Usability.** Effort required to learn, operate, prepare input for, and interpret output of a program.
5. **Maintainability.** Effort required to locate and fix an error in a program
6. **Flexibility.** Effort required to modify an operational program.

7. **Testability**. Effort required to test a program to ensure that it performs its intended function.

8. **Portability**. Effort required to transfer the program from one hardware and/or software system environment to another.

9. **Reusability**. Extent to which a program [or parts of a program] can be reused in other applications.

10. **Interoperability**. Effort required to couple one system to another.



# ISO 9126 Quality Factors

- The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software. The standard identifies six key quality attributes:
- ➤ **Functionality.** The degree to which the software satisfies stated needs as indicated by the following sub attributes: suitability, accuracy, interoperability, compliance, and security.
- ➤ **Reliability.** The amount of time that the software is available for use as indicated by the following sub attributes: maturity, fault tolerance, recoverability.
- **Usability.** The degree to which the software is easy to use as indicated by the following sub attributes: understandability, learnability, operability.

- **Efficiency.** The degree to which the software makes optimal use of system resources as indicated by the following sub attributes: time behavior, resource behavior.
- **Maintainability.** The ease with which repair may be made to the software as indicated by the following sub attributes-analysability, changeability, stability, testability.
- **Portability.** The ease with which the software can be transposed from one environment to another as indicated by the following sub attributes: adaptability, installability, conformance, replaceability

# Targeted Quality Factors

- ➤ **Intuitiveness**. A beginner, can use the interface without needing much training.
- ➤ **Efficiency**: How easily you can find and start using operations and information
- ➤ **Robustness**. The degree to which the software handles bad input data or inappropriate user interaction.
- ➤ **Richness**. The degree to which the interface provides a rich feature set.

# SOFTWARE QUALITY DILEMMA

## 1. Good Enough Software:

- Decent software provides the features users want, but it also has some lesser-known features with known issues (bugs). The hope is that most users will be so satisfied with the main features that they won't mind the bugs in the less-used ones.

## 2. The Cost of Quality:

The cost of quality includes all the expenses associated with ensuring **good quality and fixing problems** caused by poor quality.

The cost of quality can be divided into costs associated with **prevention, appraisal, and failure**.

- **Prevention costs** include
  - (1) the cost of management activities required to plan and coordinate all quality control and quality assurance activities,
  - (2) the cost of added technical activities to develop complete requirements and design models,
  - (3) test planning costs, and
  - (4) the cost of all training associated with these activities.

- **Appraisal costs** include
  - (1) the cost of conducting technical reviews for software engineering work products
  - (2) the cost of data collection and metrics evaluation, and
  - (3) the cost of testing and debugging.
- **Failure costs** are those that would disappear if no errors appeared before shipping a product to customers.

# ACHIEVING SOFTWARE QUALITY

- To achieve high software quality: software engineering methods, project management techniques, quality control actions, and software quality assurance.
- 1. **Software Engineering Methods:** If you expect to build high-quality software, you must understand the problem to be solved. You must also be capable of creating a design that conforms to the problem while at the same time exhibiting characteristics that lead to software that exhibits the quality dimensions and factors.

2. **Project Management Techniques:** (1) A project manager uses estimation to verify that delivery dates are achievable, (2) schedule dependencies are understood and the team resists the temptation to use shortcuts, (3) risk planning is conducted so problems do not appear.
3. **Quality Control:** Quality control encompasses a set of software engineering actions that help to ensure that each work product meets its quality goals. Models are reviewed to ensure that they are **complete and consistent**. Code may be inspected in order to **uncover and correct errors** before testing commences.



# ELEMENTS OF SOFTWARE QUALITY ASSURANCE

- Software quality assurance encompasses a broad range of concerns and activities that focus on the management of software quality.
1. **Standards.** The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.
  2. **Reviews and audits.** 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.

**3. Testing.** 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-finding error.

**4. 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 used to eliminating them.

**5. Change management.** If change 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.

**6. 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 a key proponent and sponsor of educational programs.

**7. 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 Web Apps, and ensure that software has not been tampered with internally. **SQA ensures that appropriate process and technology are used to achieve software security.**

**8. 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.**

9. **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/Activities

- SQA activities that address
  - 1. quality assurance planning,
  - 2. oversight,
  - 3. record keeping,
  - 4. analysis, and
  - 5. reporting.

- **Quality Assurance Planning:**
  - This involves creating a comprehensive plan that outlines how quality will be ensured throughout the software development process. It includes defining quality standards, methodologies, processes, and the resources needed to achieve and maintain the desired level of quality.
- **Oversight:**
  - Oversight in SQA refers to the continuous monitoring and supervision of the software development process to ensure that the product established quality standards and follows the planned procedures. This activity involves regular checks and reviews to identify and address any deviations from the quality assurance plan.

- **Record Keeping:**

- Record keeping involves maintaining **detailed documentation** of various aspects of the software development process. This includes **records of quality standards, test results, reviews, changes made, and any issues encountered**. Accurate record keeping is crucial for analysis, auditing, and maintaining a historical perspective on the project.

- **Analysis:**

- Analysis in SQA involves **evaluating the data and information collected during the software development process**. This can include analyzing test results, reviewing documentation, and assessing adherence to quality standards.



- **Reporting:**
- Reporting involves communicating the results of quality assurance activities to relevant stakeholders. This includes creating and distributing reports on the status of quality, highlighting any issues or concerns, and providing insights gained from the analysis.

# SQA GOALS

- **Requirement's quality.** SQA must ensure that the software team has properly reviewed the requirements model to achieve a high level of quality.
- **Design quality.** Every element of the design model should be assessed by the software team to ensure that it exhibits high quality

- **Quality control effectiveness.** SQA analyzes the allocation of resources for reviews and testing to assess whether they are being **allocated in the most effective manner.**
- **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.**

# Software Process Improvement (SPI)

- Software Process Improvement (SPI) methodology is defined as a **sequence of tasks, tools, and techniques to plan and implement improvement activities to achieve specific goals such as increasing development speed, achieving higher product quality or reducing costs**
- The SPI strategy transforms the existing approach to software development into something that is more focused, more repeatable, and more reliable.

# SPI Framework

- **Characteristics for Effective Software Process:**
  - This means there are specific features or qualities that need to be in place for a software development process to work well. For example, clear communication, thorough testing, and proper documentation.
- **Assessment Method:**
  - This is like a way to check if those important features are actually happening in the software development process. It's a method or a set of steps to see if things are going the way they should.

- **Results Summary Mechanism:**

- After checking, you need a way to give a quick overview of what you found. Think of it like a report card for a software development process. It summarizes how well things are going.

- **Implementation Assistance Strategy:**

- If you find some areas are not working well, you need a plan to help improve them. This is the strategy part. It's like having a game plan for making things better in the areas where they are not up to the mark

# SPI PROCESS

## 1. Assessment and Gap Analysis:

- Helps you understand how well your organization follows the current software process and practices. It looks for both strengths and weaknesses, highlighting areas where there's room for improvement.
- The gap between what's currently done and the best practices reveals opportunities for enhancement.

## 2. Education and Training

For practitioners, technical managers, and more senior. Three types of education and training should be conducted: generic software engineering concepts and methods, specific technology and tools, and communication and quality-oriented topics.

## 3. Selection and Justification

Pick a process model that suits your organization, stakeholders, and the software you're creating. Decide on the key activities, the main things you'll create, and quality checkpoints to track progress. If a review shows weaknesses (like not having a formal quality assurance process), concentrate on improving those specific areas directly.



#### 4. Installation and Migration

When you make changes to improve your software development process the first impact is felt during installation. This involves defining and implementing new activities, actions, and tasks as part of a fresh software development approach. These changes are big and affect both the organization and the technology used, so they need careful management.

Changes associated with SPI are relatively minor, representing small, but meaningful modifications to an existing process model. Such changes are often referred to as *process migration*. An incremental *migration* from one process (that doesn't work as well as desired) to another process is a more effective strategy for improvement.

## 5. Evaluation

- The evaluation activity 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.

## 6. Risk Management

- A software organization should manage risk at three key points in the SPI process
  1. *prior to the initiation of the SPI frame work*
  2. *during the execution of SPI activities (assessment, education, selection, installation)*
  3. *during the evaluation activity.*

# SPI risk factors

- Budget and cost,
- Content and deliverables
- Maintenance of SPI deliverables,
- Organizational management
- Organizational stability
- Process stakeholders
- Schedule for SPI development
- SPI development environment
- SPI development process
- SPI project management
- SPI staff

# CMMI MODEL

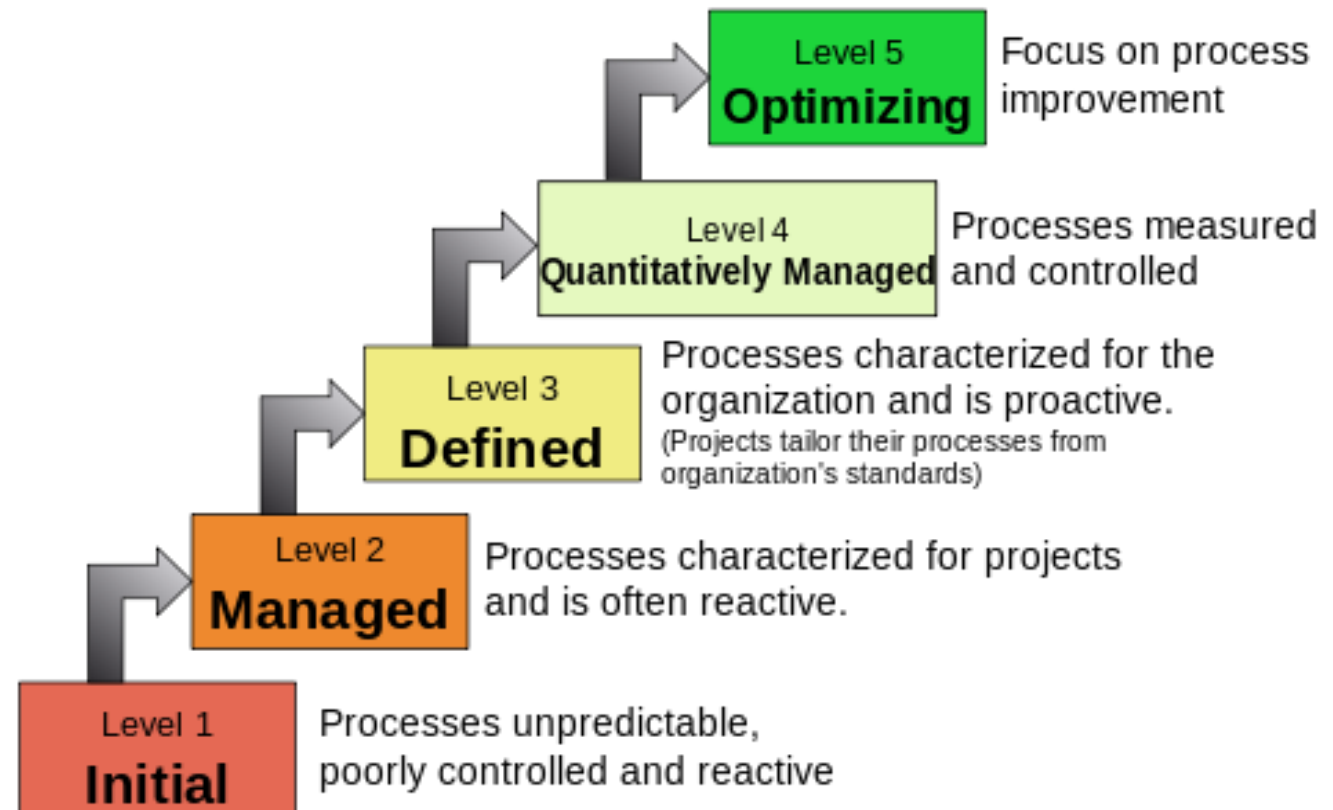
## *Capability Maturity Model Integration (CMMI)*

- Is a framework that defines the maturity of an organization's software development processes.
- It provides **a set of guidelines and best practices to assess and improve the software development process**
- The model is structured into maturity levels, ranging from an initial, ad-hoc stage to an optimized, well-defined stage, helping organizations enhance their software development capabilities over time

# Maturity Level

- **Initial: Level 1**
  - Processes are unpredictable, poorly controlled, and reactive. There's a lack of systematic management.
- **Managed: Level 2**
  - Basic project management processes are established to track cost, schedule, and functionality. There's an increasing level of control.
- **Defined: Level 3**
  - Processes are well characterized and understood. Standards are defined and used throughout the organization.
- **Quantitatively Managed: Level 4**
  - Detailed measures of the process and product quality are collected.
- **Optimizing: Level 5**
  - Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

## Characteristics of the Maturity levels



# Capability level

- **Capability level 0 : Incomplete**
  - incomplete process – partially or not performed.
  - No generic goals are specified for this level.
  - this capability level is same as maturity level 1.
- **Capability level 1 : Performed**
  - process you're following may not be perfect or meet all the goals like quality, cost, and schedule, it's still getting some work done.
  - it's like taking a small step towards improving the process. It's a beginning, but not a guarantee of success.
- **Capability level 2 : Managed**
  - process is planned, monitored and controlled.
  - managing the process by ensuring that objectives are achieved.
  - objectives are both model and other including cost, quality, schedule.
  - actively managing processing with the help of metrics.



- **Capability level 3 : Defined**

- a defined process is managed and meets the organization's set of guidelines and standards.
- focus is process standardization.

- **Capability level 4 : Quantitatively Managed**

- process is controlled using statistical and quantitative techniques.
- process performance and quality is understood in statistical terms and metrics.
- quantitative objectives for process quality and performance are established.

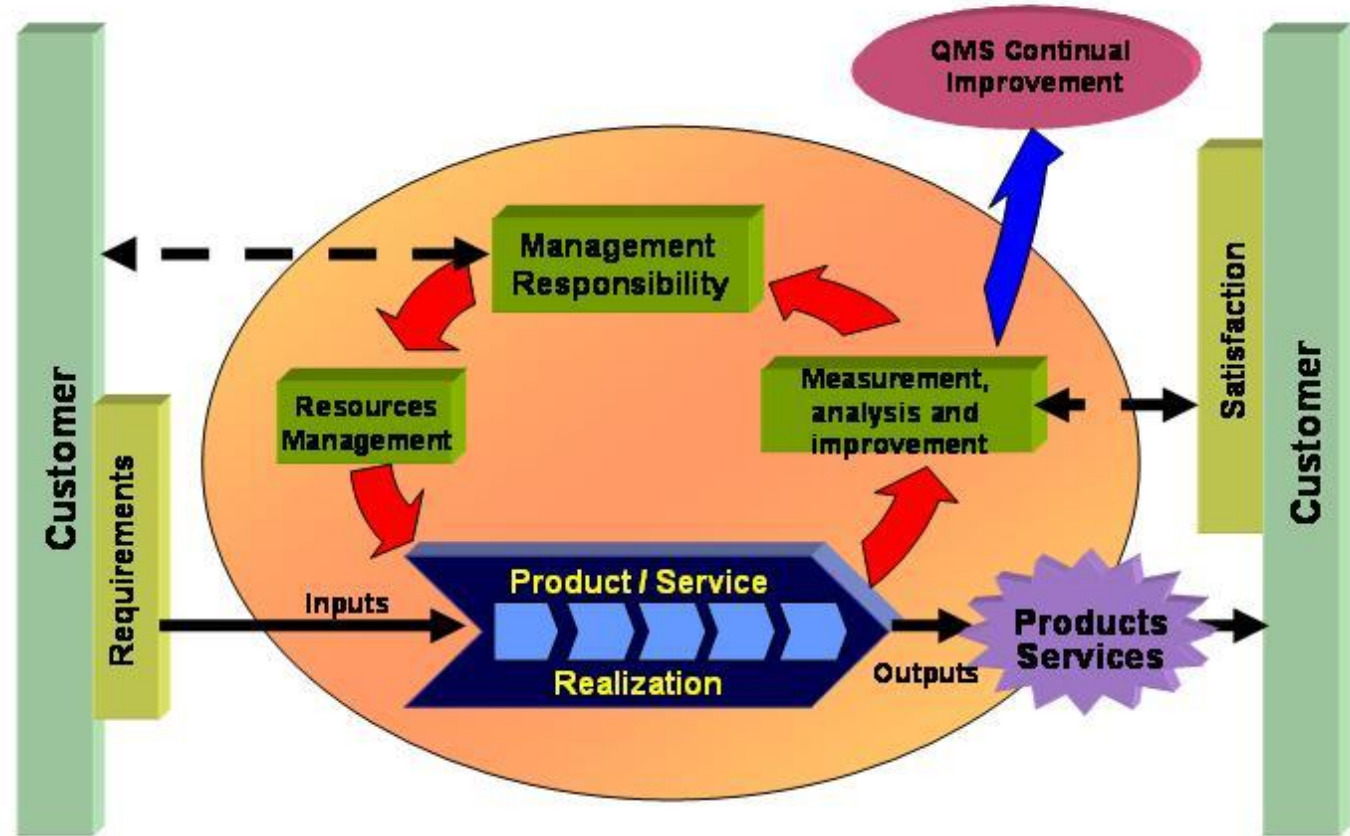
- **Capability level 5 : Optimizing**


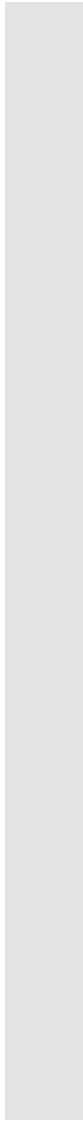
- focuses on continually improving process performance.
- performance is improved in both ways – incremental and innovation.
- emphasizes on studying the performance results across the organization to ensure that common causes or issues are identified and fixed.

# Goal of CMMI model

- **Improve Process Efficiency:**
  - Enhance the efficiency of an organization's processes for developing and delivering products or services.
- **Ensure Quality:**
  - Establish and maintain high-quality standards in all aspects of the organization's work.
- **Increase Productivity:**
  - Boost productivity by implementing best practices and optimizing processes.
- **Enhance Customer Satisfaction:**
  - Improve customer satisfaction by delivering products and services that meet or exceed customer expectations.
- **Facilitate Continuous Improvement:**
  - Foster a culture of continuous improvement, where processes are regularly assessed and enhanced over time.

# ISO 9001:2000 for software



- 
- 
- ISO 9001:2000 refers the standard sets out the criteria for establishing, implementing, maintaining, and continually improving a quality management system within an organization.
  - The ISO 9001:2000 version was later revised and replaced by ISO 9001:2008, and subsequently by ISO 9001:2015. The revisions reflect updates and improvements in the understanding and practice of quality management systems.

# key aspects of ISO 9001:2000

- **Process Approach:** ISO 9001:2000 introduced a process approach to quality management, emphasizing the identification and management of interrelated processes within an organization.
- **Customer Focus:** The standard places a strong emphasis on meeting customer requirements and enhancing customer satisfaction.
- **Continuous Improvement:** ISO 9001:2000 emphasizes the concept of continual improvement, encouraging organizations to regularly review and improve their processes and quality management system.
- **Documentation Requirements:** The standard outlines documentation for a quality management system, including a quality manual, documented procedures, and records.

# 8 principles

- **Customer-focused organization:** organizations should understand customers' current and future needs, and exceed their expectations.
- **Leadership:** establish a unity of purpose and direction creating an internal environment where people can contribute to achieve the organization's expected results.
- **Involvement of people:** full involvement of employees enables their abilities to be used for the organization's benefit.
- **Process approach:** systematic identification and management of the various processes employed within an organization and the interactions among these processes in order to obtain the desired result.

- **Systemic approach:** managing a system of interrelated processes to a given objective contributes to efficiency. Integrating and aligning processes leads to better results.
- **Continual improvement:** should be a permanent objective of the organization, leading to improvements in the overall performance.
- **Factual approach to decision making:** effective decisions are based on the logical analysis of reliable data and information
- **Mutually beneficial customer relationships:** establishing relationships with customer to enhance the ability of both organizations to create value.

# Goals

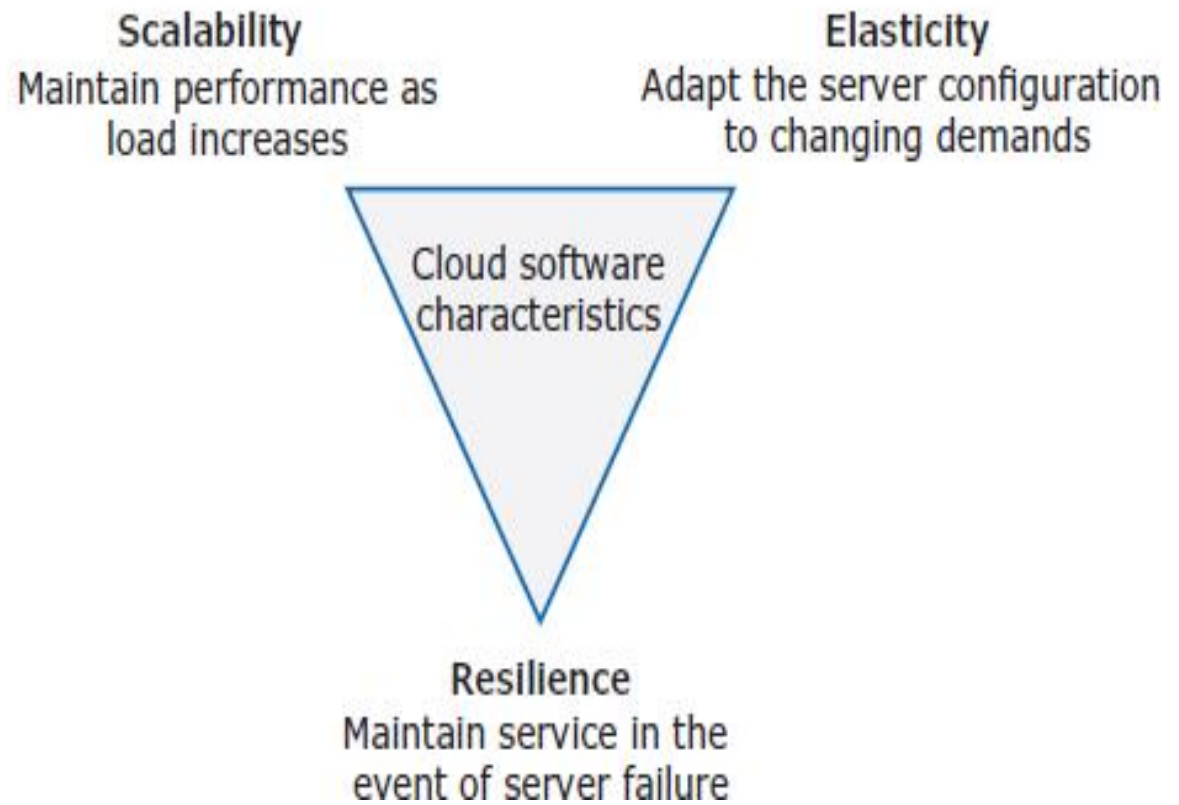
- More effective management of related processes Process standardization
- Better control of documents and records
- Cost reduction
- Effective monitoring of processes
- Focus on customer satisfaction enhancement
- Continual improvement



# CLOUD BASED SOFTWARE

- 
- Cloud-based computing (also called Software as a Service, or SaaS) **allows users access to software applications that run on shared computing resources (for example, processing power, memory, and disk storage) via the Internet.**
- These computing resources are maintained in remote data centers dedicated to hosting various applications on multiple platforms.

**Figure 5.1** Scalability, elasticity, and resilience



## *Characteristics of Cloud based Software:*

- **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.**
- Systems can be scaled by adding new servers or by migrating to a more powerful server.

- **Elasticity**, you can monitor the demand on your application and add or remove servers dynamically as the number of users changes. This means that you pay for only the servers you need when you need them.
- **Resilience** means that you can design your software architecture to tolerate server failures. You can make several copies of your software available concurrently. If one of these fails, the others continue to provide a service.

# VIRTUALIZATION AND CONTAINERS

- Virtual server made up of an operating system plus a set of software 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.
- 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
- The problem with implementing virtual servers in VM 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 provider. This means that you cannot instantly react to changing demands by starting up and shutting down VMs.

# *Container Based Virtualization*

- Using containers dramatically speeds up the process of deploying virtual servers on the cloud.
- Containers are usually megabytes in size, whereas VMs are gigabytes.
- Containers can be started up 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.**
- Containers are a lightweight mechanism for running applications in the cloud and are particularly effective for running small applications such as stand-alone services

- From a cloud software engineering perspective, containers offer four important benefits:

1.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.

2.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.

3.They provide an efficient mechanism for implementing software services.

4.Simplified adoption of DevOps. This is an approach to software support where the same team is responsible for both developing and supporting operational software

# IaaS, PaaS, SaaS

## Infrastructure as a service (IaaS)

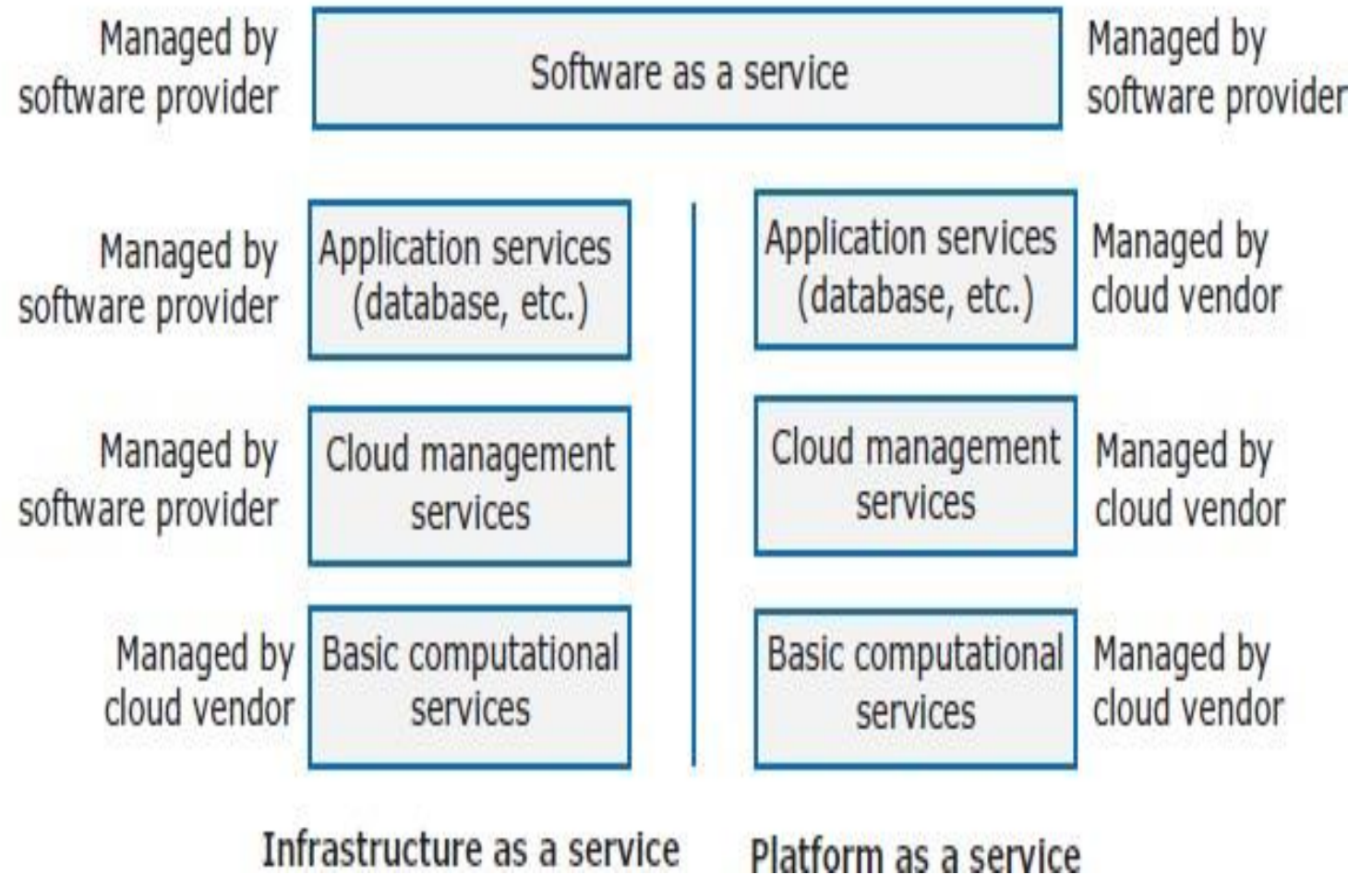
- This is a basic service level that all major cloud providers offer.
- They provide different kinds of infrastructure service, such as a **compute service, a network service, and a storage service**.
- These infrastructure services may be used to implement **virtual cloud-based servers**.
- The key 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, including SQL and NoSQL databases.
- Using PaaS makes it easy to develop auto-scaling software.
- You can implement your product so that as the load increases, additional compute and storage resources are added automatically.

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

**Figure 5.6** Management responsibilities for SaaS, IaaS, and PaaS



- An important difference between IaaS and PaaS is the allocation of system management responsibilities. Figure 5.6 shows who has management responsibilities for SaaS, IaaS, and PaaS.
- If you are using IaaS, you have the responsibility for installing and managing the database, the system security, and the application.
- If you use PaaS, you can devolve responsibility of managing the database and security to the cloud provider.
- In SaaS, assuming that a software vendor is running the system on a cloud, the software vendor manages the application. Everything else is the cloud provider's responsibility

# MICRO SERVICE ARCHITECTU RE


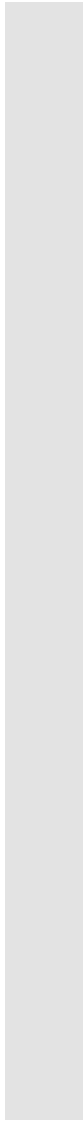
## MICROSERVICES

- Microservices, also known as the microservices architecture, is an approach to **software development where a large application is broken down into smaller, independent services.**
- Each of these services, called microservices, represents a **specific business capability and operates as a separate, self-contained unit.**
- Software products that use microservices are said to have a microservices architecture.
-

# Characteristics of Microservices:

- **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)
- **Fault Isolation**: If one microservice fails, it doesn't necessarily affect the entire application. The impact is limited to the specific service, promoting fault isolation.
- **Single Responsibility**: Each microservice is designed to perform a specific business function or service. It follows the principle of "single responsibility," focusing on doing one thing well.

- Cohesion and coupling are ideas that were developed in the 1970s to reflect the interdependence of components in a software system. Briefly:
  - Coupling is a measure of the number of relationships that one component has with other components in the system. Low coupling means that components do not have many relationships with other components.
- - Cohesion is a measure of the number of relationships that parts of a component have with each other. High cohesion means that all of the component parts that are needed to deliver the component's functionality are included in the component.

- 
- 
- Low coupling is important in microservices because it leads to independent services, you can update a service without having to change other services in the system.
  - High cohesion is important because it means that the service does not have to call lots of other services during execution. Calling other services involves communications overhead, which can slow down a system.



- Message management code in a microservice is responsible for processing incoming and outgoing messages. Incoming messages have to be checked for validity and the message format. Outgoing messages have to be packed into the correct format for service communication.
- Failure management code in a microservice has two concerns. First, it has to cope with circumstances where the microservice cannot properly complete a requested operation. Second, if external interactions are required, such as a call to another service, it has to handle the situation where that interaction does not succeed because the external service returns an error or does not reply
- Data consistency management is needed when the data used in a microservice are also used by other services. In those cases, there needs to be a way of communicating data updates between services and ensuring that the changes made in one service are reflected in all services that use the data.

# MICROSERVICES ARCHITECTURE

- A microservices architecture is not like a layered application architecture that defines the common set of components used in all applications of a particular kind.
- Rather, a microservices architecture is an *architectural style*—a tried and tested way of implementing a logical software architecture.

# Key Aspects

- Architecture Design Decisions
- In a microservice-based system, the development teams for each service are autonomous. They make their own decisions about how best to provide the service. This means the system architect should not make technology decisions for individual services; these are left to the service implementation team.

- Key Design Questions

- What are the microservices that makes the system?
- How should microservices communicate with each other?
- How should service failure be detected, reported and managed?
- How should the microservices in system be coordinated?
- - How should data be distributed and shared?

- General Guidelines to decompose microservices

- Balance fine-grain functionality and system performance
- Follow the “common closure principle”
- Associate services with business capabilities
- Design services so that they have access to only the data that they need

- Service Communications
- Services communicate by exchanging messages. These messages include information about the originator of the message as well as the data that are the input to or output from the request. The messages that are exchanged are structured to follow a message protocol. This is a definition of what must be included in each message and how each component of the message can be identified. When you are designing a microservices architecture, you have to establish a standard for communications that all microservices should follow. Key decisions that you have to make are:
  - Should service interaction be synchronous or asynchronous?
  - Should services communicate directly or via message broker middleware?
  - What protocol should be used for messages exchanged between services?

- *Data Distributions and Sharing*
- A general rule of microservice development is that each microservice should manage its own data. You need to think about the microservices as an interacting system rather than as individual units. This means:
- You should isolate data within each system service with as little data sharing as possible.
- If data sharing is unavoidable, you should design microservices so that most sharing is read-only, with a minimal number of services responsible for data updates.
- If services are replicated in your system, you must include a mechanism that can keep the database copies used by replica services consistent.


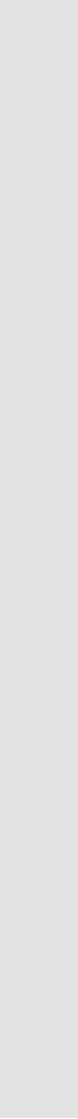
- Service Coordination
- Most user sessions involve a series of interactions in which operations have to be carried out in a specific order. This is called a workflow.
- An alternative approach that is often recommended for microservices is called “choreography.” This term is derived from dance rather than music, where there is no “conductor” for the dancers. Rather, the dance proceeds as dancers observe one another. Their decision to move on to the next part of the dance depends on what the other dancers are doing.

- In a microservices architecture, choreography depends on each service emitting an event to indicate that it has completed its processing. Other services watch for events and react accordingly when events are observed. There is no explicit service controller.
- 
- To implement service choreography, you need additional software such as a message broker that supports a publish
- and subscribe mechanism. Publish and subscribe means that services “publish” events to other services and “subscribe” to those events that they can process.



# *Failure Management*

Failure type	Explanation
Internal service failure	These are conditions that are detected by the service and can be reported to the service requestor in an error message. An example of this type of failure is a service that takes a URL as an input and discovers that this is an invalid link.
External service failure	These failures have an external cause that affects the availability of a service. Failure may cause the service to become unresponsive and actions have to be taken to restart the service.
Service performance failure	The performance of the service degrades to an unacceptable level. This may be due to a heavy load or an internal problem with the service. External service monitoring can be used to detect performance failures and unresponsive services.

- 
- 
- The simplest way to report microservice failures is to use HTTP status codes, which indicate whether or not a request has succeeded. Service responses should include a status that reflects the success or otherwise of the service request. Status code 200 means the request has been successful, and codes from 300 to 500 indicate some kind of service failure. Requests that have been successfully processed by a service should always return the 200 status code.

# MICROSERVICE DEPLOYMENT

- A general principle of micro service-based development is that the service 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. Continuous deployment means that as soon as a change to a service has been made and validated, the modified service is re- deployed.
- Continuous deployment is a process in which new versions of a service are put into production as soon as a service change has been made. It is a completely automated process that relies on automated testing to check that the new version is of production quality.
- If continuous deployment is used, you may need to maintain multiple versions of deployed services so that you can switch to an older version if problems are discovered in a newly deployed service
-

- The deployment of a new service version starts with the programmer committing the code changes to a code management system such as Git.
- Deployment involves adding the new service to a container and installing the container on a server. Automated “whole system” tests are then executed. If these system tests run successfully, the new version of the service is put into production
- Containers are usually the best way to package a cloud service for deployment. Containers are a deployment unit that can execute on different servers so that the service development team does not have to take server configuration issues into account