MANAGEMENT OF SOFTWARE SYSTEMS

Module 5

Rijin IK

Assistant Professor
Department of Computer Science and Engineering
Vimal Jyothi Engineering College
Chemperi

February 11, 2023

Outline

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**.
- The definition serves to emphasize three main points:
  - An **effective software process** establishes the infrastructure that supports any effort at building a high-quality software product.
  - A **useful product** delivers the content, functions, and features that the end user desires, but as important, it delivers these assets in a reliable, error-free way.
  - By adding value for both the **producer and user** of a software product, high-quality software provides benefits for the software organization and the end-user community.

Software Quality

Garvin's Quality Dimensions

- Performance Quality.
  - Performance refers to a **product's primary operating characteristics.**
  - **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.
  - Features are **additional characteristics that enhance the appeal of the product** or service to the user.
  - 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?

Software Quality

Garvin's Quality Dimensions cont..

- Durability.
  - **A measure of product life**
  - **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.
  - **Aesthetics refers to the appearance of a product or service**
  - Does our product have a unique style?
- Perception.
  - Meaning that **a product or service can have high scores** on each of the seven dimensions of quality, **but still receive a bad rating from customers as a result of negative perceptions from customers or the public.**

McCall's software quality factors

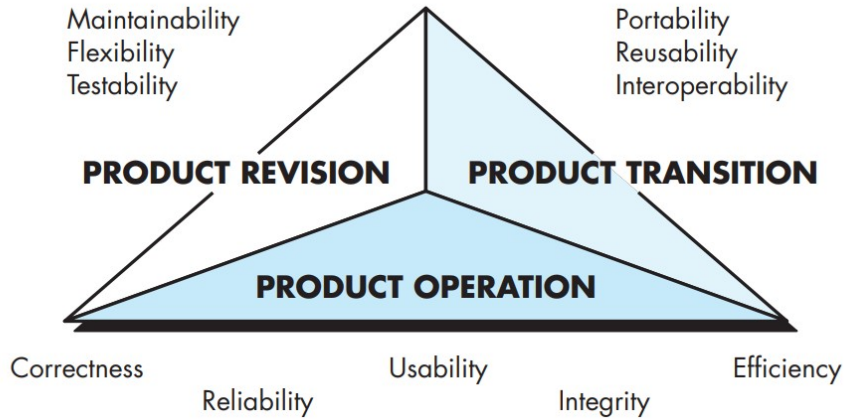

Figure 1: McCall's software quality factors

- This model classifies all software requirements into 11 software quality factors.
- The 11 factors are grouped into three categories
  - **Product operation factors:** Correctness, Reliability, Efficiency, Integrity, Usability.
    Product revision factors: Maintainability, Flexibility, Testability.
    Product transition factors: Portability, Reusability, Interoperability.

Software Quality

McCall's software quality factors cont..

- **Correctness:** The extent to which a software meets its requirements specification.
- **Reliability:**The extent to which a software performs its intended functions without failure.
- **Efficiency:** The amount of computing resources and code required by a program to perform its function.
- **Integrity:** Extent to which access to software or data by unauthorized persons can be controlled.
- **Usability:**The extent of effort required to learn, operate and understand the functions of the software.
- **Maintainability:** Effort required to locate and fix an error in a program.
- **Flexibility:** Effort required to modify an operational program.
- **Testability** Effort required to test a program to ensure that it performs its intended function.

McCall's software quality factors cont..

- **Portability:** Effort required to transfer the program from one hardware and/or software system environment to another.
- **Reusability:** Extent to which a program [or parts of a program] can be reused in other applications—related to the packaging and scope of the functions that the program performs.
- **Interoperability:** Effort required to couple one system to another.

ISO 9216 Quality Factors

- 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 behaviour.

ISO 9216 Quality Factors cont..

- Maintainability.
  - The ease with which repair may be made to the software as indicated by the following sub attributes: analyzability, 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

Software Quality Dilemma

Software Quality Dilemma cont..

- The quality dilemma is basically two fold:
  - Build bad software (low quality) and no one will want to buy/use 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 Dilemma

Software Quality Dilemma

- **"Good Enough" Software:**
  - Good enough software **delivers high quality functions and features** that end-users desire, **but at the same time** it delivers other more unclear or specialized functions and features **that contain known bugs .**
- **The Cost of Quality:**
  - Includes all costs incurred in the pursuit of quality or in performing quality-related activities and the downstream costs of lack of quality.
  - The cost of quality can be divided into costs associated with prevention, appraisal, and failure.
    - **Prevention:** management activities, training, test planning
    - **Appraisal:** technical reviews, data collection and metrics, testing and debugging
    - **Failure**
      Internal: repair work, network hiccups
      External:product return and replacement, help-line
- **Risks:**
  - The implication is that low-quality software increases risks for both the developer and the end user.

Software Quality Dilemma

Software Quality Dilemma cont..

- **Negligence and Liability:**
  - **Work begins with the best of intentions on both sides**, but **by the time the system is delivered, things have gone bad**. The system is late, **fails to deliver desired features and functions**, is error-prone, and **does not meet with customer approval**. Legal actions happens.
- **Quality and Security:**
  - software that **does not exhibit high quality is easier to hack**, and as a consequence, **low-quality** software can indirectly **increase the security risk** with all of its attendant costs and problems.
- **The Impact of Management Actions:**
  - Software **quality is often influenced as much by management decisions** as it is by technology decisions. Even the **best software engineering practices can be subverted by poor business decisions** and questionable project management actions.

Achieving Software Quality

Software quality doesn't just appear. It is the result of good project management and solid software engineering practice, Management and practice are applied within the context of **four broad activities that help a software team achieve high software quality:**

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:** Software quality can be improved if
   i. a project manager uses estimation to verify that delivery dates are achievable,
   ii. schedule dependencies are understood and the team resists the temptation to use shortcuts,
   iii. risk planning is conducted

Achieving Software Quality cont..

3 **Quality Control:**

- Quality control encompasses a set of software engineering actions that help to **ensure that each work product meets its quality goals.**
- A combination of measurement and feedback allows a software team to tune the process when any of these work products fail to meet quality goals.

4 **Quality Assurance:**

- Quality assurance (QA) is any **systematic process used to determine if a product or service meets quality standards.**
- Quality assurance establishes the infrastructure that supports solid software engineering methods, rational project management, and quality control actions—all pivotal if you intend to build high-quality software.

Software Quality Assurance

- Software quality assurance (often called quality management) is an activity that is applied throughout the software process.
- Software quality assurance (SQA) encompasses:
  1. an **SQA process**,
  2. specific **quality assurance and quality control tasks** (including technical reviews and a multi-tiered testing strategy),
  3. effective **software engineering practice** (methods and tools),
  4. control of all software **work products and the changes made to them** ,
  5. a **procedure to ensure compliance with** software development **standards** (when applicable), and
  6. **measurement and reporting mechanisms.**

Elements of Software Quality Assurance

Elements of Software Quality Assurance

1. Standards
2. Reviews and audits
3. Testing
4. Error/defect collection and analysis.
5. Change management.
6. Education.
7. Vendor management
8. Security management
9. Safety
10. Risk management

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.
- 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** performed by software engineers for 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.

Elements of Software Quality Assurance

Testing

- Software testing is a quality control function that has one primary goal – **to find errors**. The job of SQA is to **ensure that testing is properly and efficient conducted.**

Error/defect collection and analysis

- 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.**

Elements of Software Quality Assurance

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.

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.
  - **The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible)**, and incorporating quality mandates as part of any contract with an external vendor.

Elements of Software Quality Assurance

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.**

Elements of Software Quality Assurance

Risk management

- The SQA organization **ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.**

SQA Tasks

- The Software Engineering Institute recommends a **set of SQA activities that address quality assurance planning, oversight, record keeping, analysis, and reporting.**
- These activities are performed (or facilitated) by an independent SQA group that:
  - **Prepares an SQA plan** for a project.
  - **Participates in the development** of the project's software process description
  - **Reviews software engineering activities** to verify compliance with the defined software process.
  - **Audits designated software work products** to verify compliance with those defined as part of the software process.
  - **Ensures that deviations in software work** and work products **are documented and handled according to a documented procedure**
  - **Records any noncompliance** and **reports to senior management.**

SQA Tasks

Prepares an SQA plan for a project

- The plan is **developed as part of project planning** and is reviewed by all stakeholders.
- Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan.
- The plan identifies
  - evaluations to be performed,
  - audits and reviews to be conducted,
  - standards that are applicable to the project,
  - procedures for error reporting and tracking,
  - work products that are produced by the SQA group,
  - and feedback that will be provided to the software team.

SQA Tasks

Participates in the development of the project's software process description

- The software team selects a process for the work to be performed.
- 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.

SQA Tasks

Reviews software engineering activities to verify compliance with the defined software process.

- The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

Audits designated software work products to verify compliance with those defined as part of the software process

- The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.

SQA Tasks

Ensures that deviations in software work and work products are documented and handled according to a documented procedure

- Deviations may be encountered in the project plan, process description, applicable standards, or software engineering work products.

Records any noncompliance and reports to senior management

- Noncompliance items are tracked until they are resolved.

SQA Tasks

SQA GOALS

- Requirement's quality
- Design quality
- Code quality
- Quality control effectiveness

SQA GOALS

Requirement's quality

- The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow. 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 y the software team to ensure that it exhibits high quality and that the design itself conforms to requirements.

SQA GOALS

Code quality

- Source code and related work products 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. SQA analyzes the allocation of resources for reviews and testing to assess whether they are being allocated in the most effective manner.

Software measurement and metrics

Goal	Attribute	Metric
Requirement quality	Ambiguity	Number of ambiguous modifiers (e.g., many, large, human-friendly)
	Completeness	Number of TBA, TBD
	Understandability	Number of sections/subsections
	Volatility	Number of changes per requirement
		Time (by activity) when change is requested
	Traceability	Number of requirements not traceable to design/code
	Model clarity	Number of UML models
		Number of descriptive pages per model
		Number of UML errors
Design quality	Architectural integrity	Existence of architectural model
	Component completeness	Number of components that trace to architectural model
		Complexity of procedural design
	Interface complexity	Average number of pick to get to a typical function or content
		Layout appropriateness
	Patterns	Number of patterns used
Code quality	Complexity	Cyclomatic complexity
	Maintainability	Design factors (Chapter 8)
	Understandability	Percent internal comments
		Variable naming conventions
	Reusability	Percent reused components
	Documentation	Readability index
QC effectiveness	Resource allocation	Staff hour percentage per activity
	Completion rate	Actual vs. budgeted completion time
	Review effectiveness	See review metrics (Chapter 14)
	Testing effectiveness	Number of errors found and criticality
		Effort required to correct an error
		Origin of error

Figure 2: Software quality goals, attributes, and metrics

Software Process Improvement (SPI)

Software Process Improvement (SPI)

- The term software process **improvement (SPI) implies many things.**
  - First, it implies that **elements of an effective software process can be defined in an effective manner;**
  - second, that an **existing organizational approach to software development can be assessed against those elements;**
  - and third, that **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).

Software Process Improvement (SPI)

SPI framework

- An organization can choose one of the many SPI frameworks.
- An SPI framework defines:
  1. a **set of characteristics** that must be present if an effective software process is to be achieved,
  2. a **method for assessing** whether **those characteristics are present,**
  3. a **mechanism for summarizing the results** of any assessment, and
  4. a **strategy for assisting a software organization** in implementing those process characteristics that have been found to be weak or missing.

Software Process Improvement (SPI)

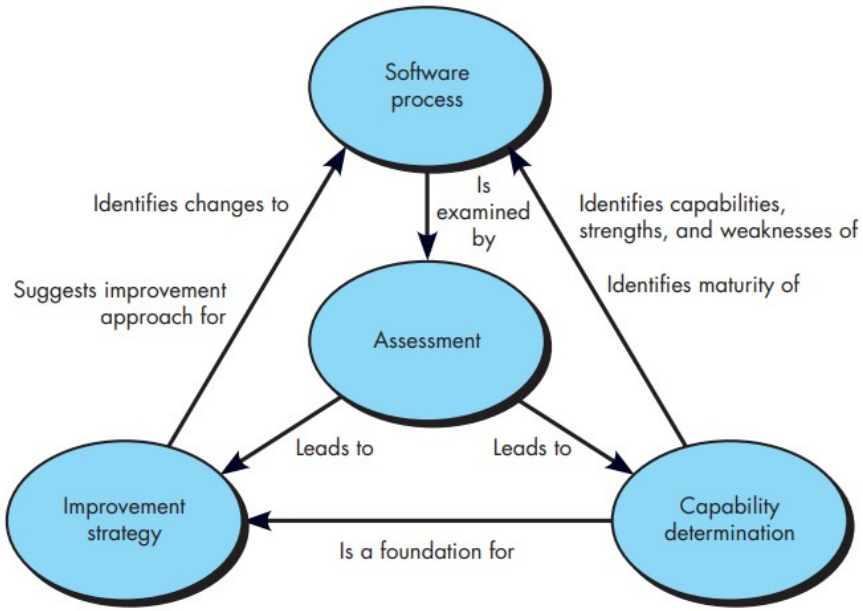

Figure 3: Elements of an SPI framework

Software Process Improvement (SPI)

SPI Process

1. Assessment and Gap Analysis
2. Education and Training
3. Selection and Justification
4. Installation/Migration
5. Evaluation
6. Risk Management for SPI

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.

Software Process Improvement (SPI)

Education and Training

- It follows that a **key element of any SPI** strategy **is education and training** for practitioners, technical managers, and more senior managers who have direct contact with the software organization.
- **Three types of education and training** should be conducted:
  - generic software engineering concepts and methods
  - specific technology and tools
  - communication and quality-oriented topics.

Software Process Improvement (SPI)

Selection and Justification

- **Choose the process model 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.

Software Process Improvement (SPI)

Installation/Migration

- **Installation** is the first point at which a software organization feels **the effects of changes implemented** as a consequence of the SPI road map. **In some cases, an entirely new process is recommended for an organization.**
- **In other cases, 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**.
- Installation and migration are actually **software process redesign (SPR)** activities.
- When a formal approach to SPR is initiated, three different process models are considered:
  - the existing ("as-is") process, ?
  - a transitional ("here-to-there") process,
  - The target ("to be") process.

Software Process Improvement (SPI)

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.
- **The overall status of the process** and the organizational culture as SPI activities proceed

Risk Management for SPI

- SPI is a risky undertaking. In fact, more than half of all SPI efforts end in failure. The reasons for failure vary greatly and are organizationally specific. Among the most **common risks are:**
  1. a lack of management support,
  2. cultural resistance by technical staff,
  3. a poorly planned SPI strategy,
  4. an overly formal approach to SPI,
  5. selection of an inappropriate process,
  6. a lack of buy-in by key stakeholders,
  7. an inadequate budget,
  8. a lack of staff training,
  9. organizational instability, and
  10. a myriad of other factors.
- The role of those chartered with the responsibility for SPI is to analyze likely risks and develop an internal strategy for mitigating them.

CMMI process improvement framework

CMMI process improvement framework

- **Capability Maturity Model Integration**
- The Capability Maturity Model Integration (CMMI) **helps organizations** streamline **process improvement, encouraging a productive, efficient culture that decreases risks** in software, product, and service development.
- Each process area (e.g., project planning or requirements management) is formally assessed against specific goals and practices and is **rated according to the following capability levels:**
  1. Level 0: Incomplete
  2. Level 1: Performed
  3. Level 2: Managed
  4. Level 3: Defined
  5. Level 4: Quantitatively managed
  6. Level 5: Optimized

CMMI process improvement framework

CMMI process improvement framework

- **Level 0: Incomplete**
  - incomplete process – partially or not performed.
  - one or more specific goals of process area are not met.
  - No generic goals are specified for this level.
  - this capability level is same as maturity level 1.
- **Level 1: Performed**
  - All of the specific goals of the process area (as defined by the CMMI) have been satisfied.
  - Work tasks required to produce defined work products are being conducted.
  - Process performance may not be stable.

CMMI process improvement framework

- **Level 2: Managed**
  - All capability level 1 criteria have been satisfied.
  - In addition, all work associated with the process area conforms to an organizationally defined policy;
  - all people doing the work have access to adequate resources to get the job done;
  - stakeholders are actively involved in the process area as required;
  - all work tasks and work products are "monitored, controlled, and reviewed;
  - and are evaluated for adherence to the process description"
- **Level 3: Defined**
  - All capability level 2 criteria have been achieved.
  - a defined process is managed and meets the organization's set of guidelines and standards.
  - focus is process standardization.

CMMI process improvement framework

CMMI process improvement framework

- **Level 4: Quantitatively managed**
  - All capability level 3 criteria have been achieved.
  - In addition, the process area is controlled and improved using measurement and quantitative assessment.
  - "Quantitative objectives for quality and process performance are established and used as criteria in managing the process".
- **Level 5: Optimized**
  - All capability level 4 criteria have been achieved.
  - In addition, the process area is adapted and optimized using quantitative (statistical) means to meet changing customer needs and to continually improve the efficacy of the process area under consideration.

ISO 9001:2000 for Software

- 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**.
- The underlying strategy suggested by ISO 9001:2000 is described in the following manner
  - 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 **assessed** through **internal or external review** processes and be evaluated on a maturity scale.
- ISO 9001:2008 **has adopted a "plan-do-check-act" cycle** that is applied to the quality management elements of a software project.

ISO 9001:2000 for Software

ISO 9001:2000 for Software cont..

- Within a software context, **"plan" establishes the process objectives, activities, and tasks necessary to achieve high-quality software** and resultant customer satisfaction.
- **"Do" implements the software process** (including both framework and umbrella activities).
- **"Check" monitors and measures the process** to ensure that all requirements established for quality management have been achieved.
- **"Act" initiates software process** improvement **activities that continually work to improve the process.**
- TickIt can be used throughout the "plan-do-check-act" cycle to ensure that SPI progress is being made.
- TickIT auditors assess the application of the cycle as a precursor to ISO 9001:2008 certification.
- For a detailed discussion of ISO 9001:2008 and TickIT you should examine [Ant06], [Tri05], or [Sch03].

Cloud-based Software

- 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 that are available on the cloud.
- The remote servers are "virtual servers," which means they are implemented in software rather than hardware.
- The cloud servers that you rent can be started up and shut down as demand changes.
- Software that runs on the cloud can be scalable, elastic, and resilient.

Cloud-based Software

- Software that runs on the cloud can be scalable, elastic, and resilient.

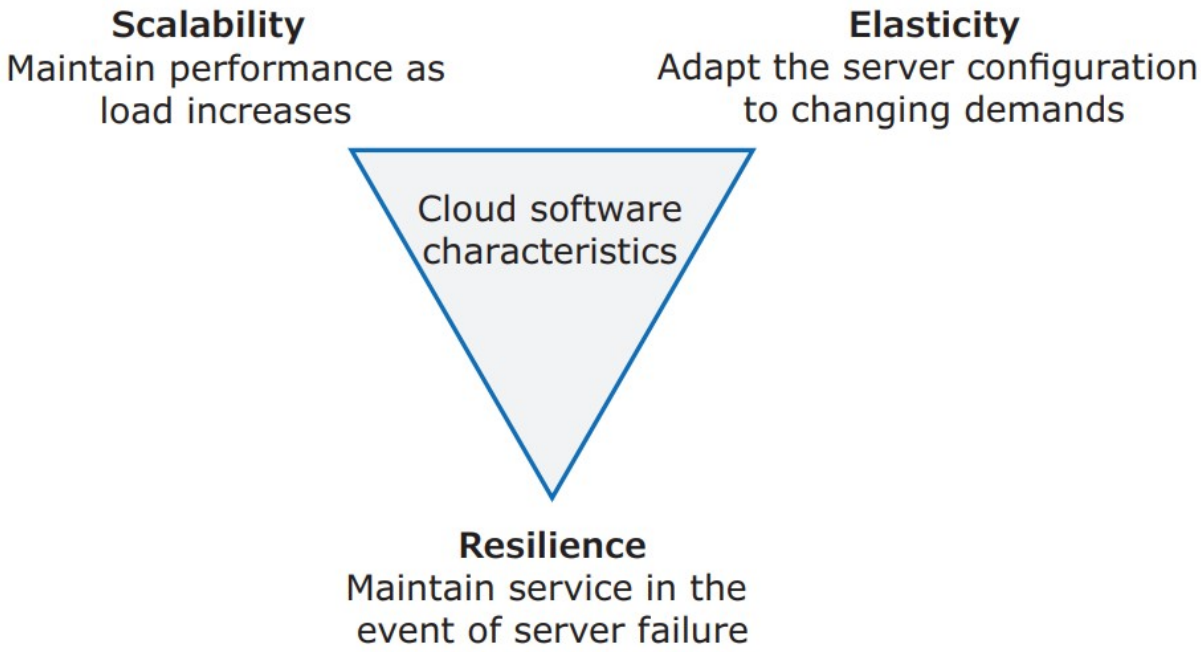

Figure 4: Scalability, elasticity, and resilience

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.

Cloud-based Software

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 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.
- This "run anywhere" characteristic is possible because the virtual server has no external dependencies.
  - An external dependency means you need some software, such as a database management system, that you are not developing yourself.
- **Virtual machines (VMs)**, running on physical server hardware, can be used to implement virtual servers.

Cloud-based Software

Virtualization and containers cont..

- **Hypervisor**
- **Container**

Hypervisor

- Also known as a **virtual machine monitor or VMM,** is software that creates and runs virtual machines (VMs).
- Hypervisor **providing a hardware emulation** that simulates the operation of the underlying hardware.
- A hypervisor allows **one host computer to support multiple guest** VMs by virtually sharing its resources, such as memory and processing.
- Several of these hardware emulators share the physical hardware and run in parallel.
- You can run an operating system and then install server software on each hardware emulator.

Cloud-based Software

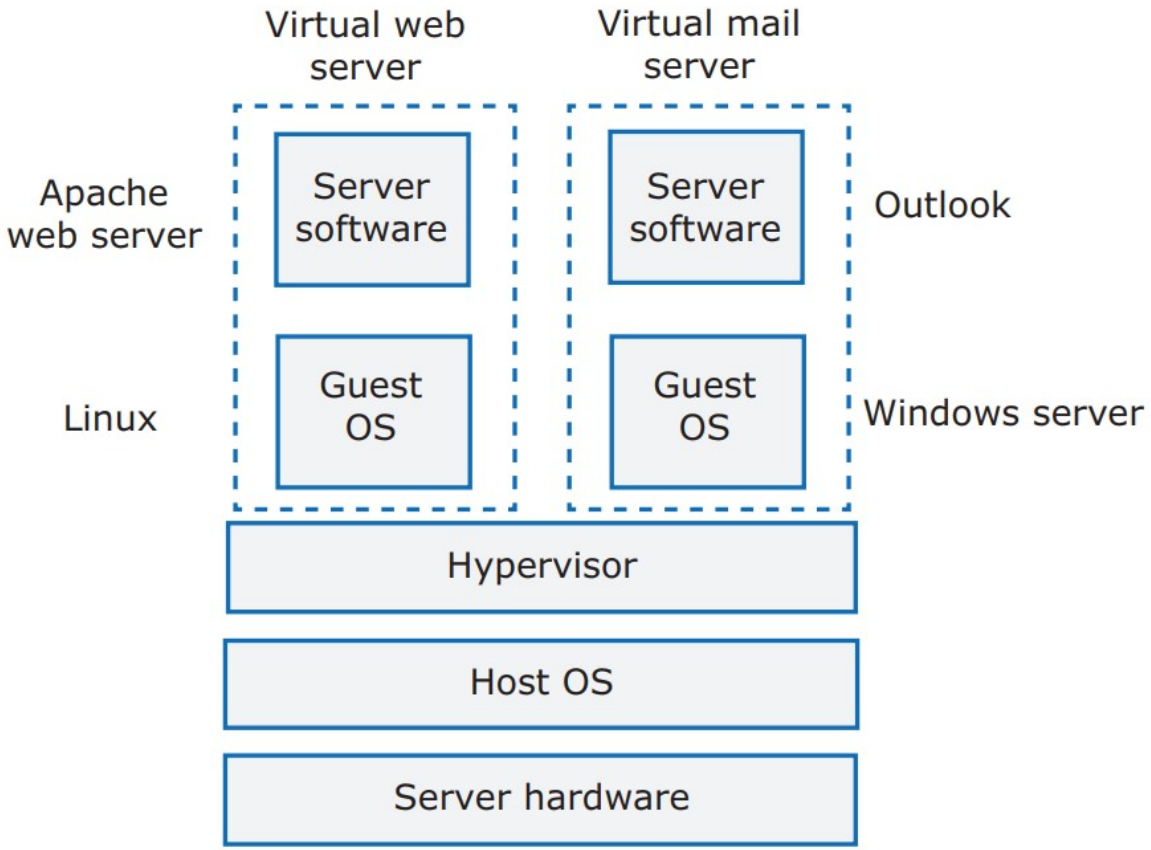

Figure 5: Implementing a virtual server as a virtual machine

Container

- Containers are an operating system virtualization technology that allows independent servers to share a single operating system.

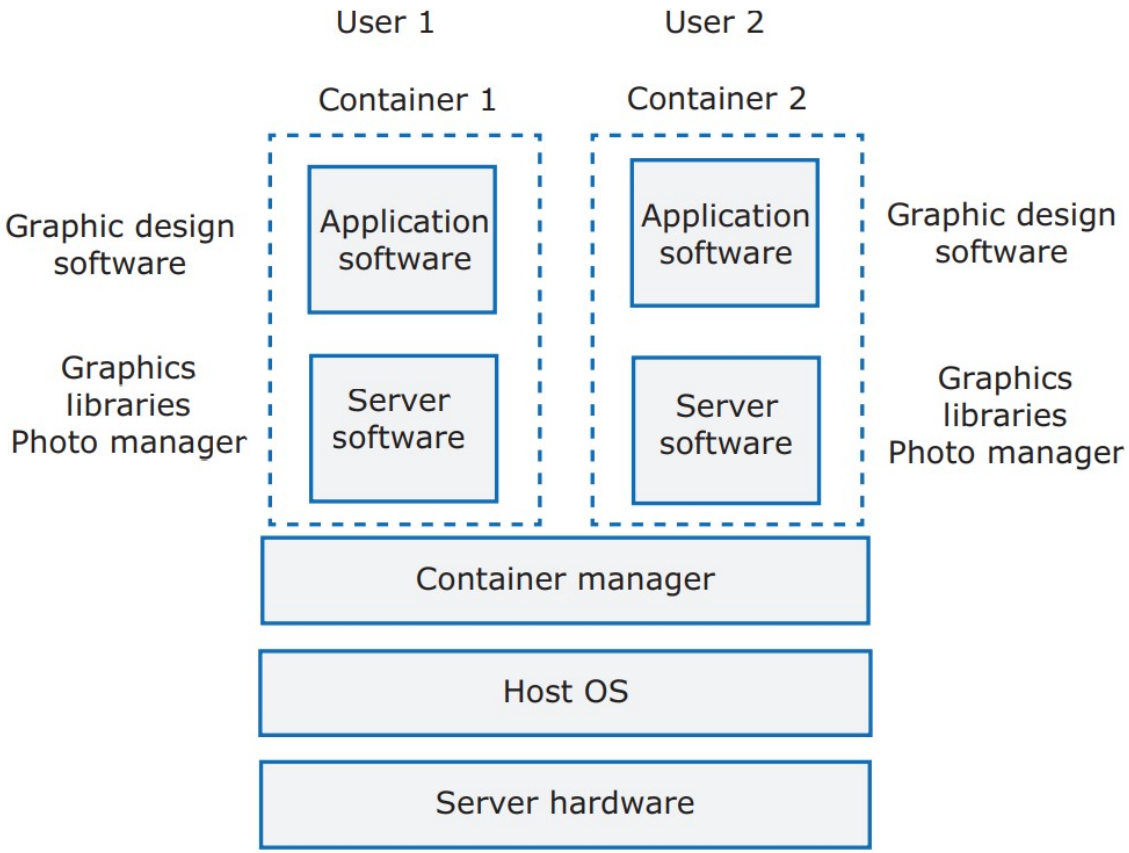

Figure 6: Using containers to provide isolated services

Advantage of using a virtual machine

- The advantage of using a virtual machine to implement virtual servers is that you have exactly the same hardware platform as a physical server.
- Therefore run different operating systems on virtual machines that are hosted on the same computer

Everything as a service

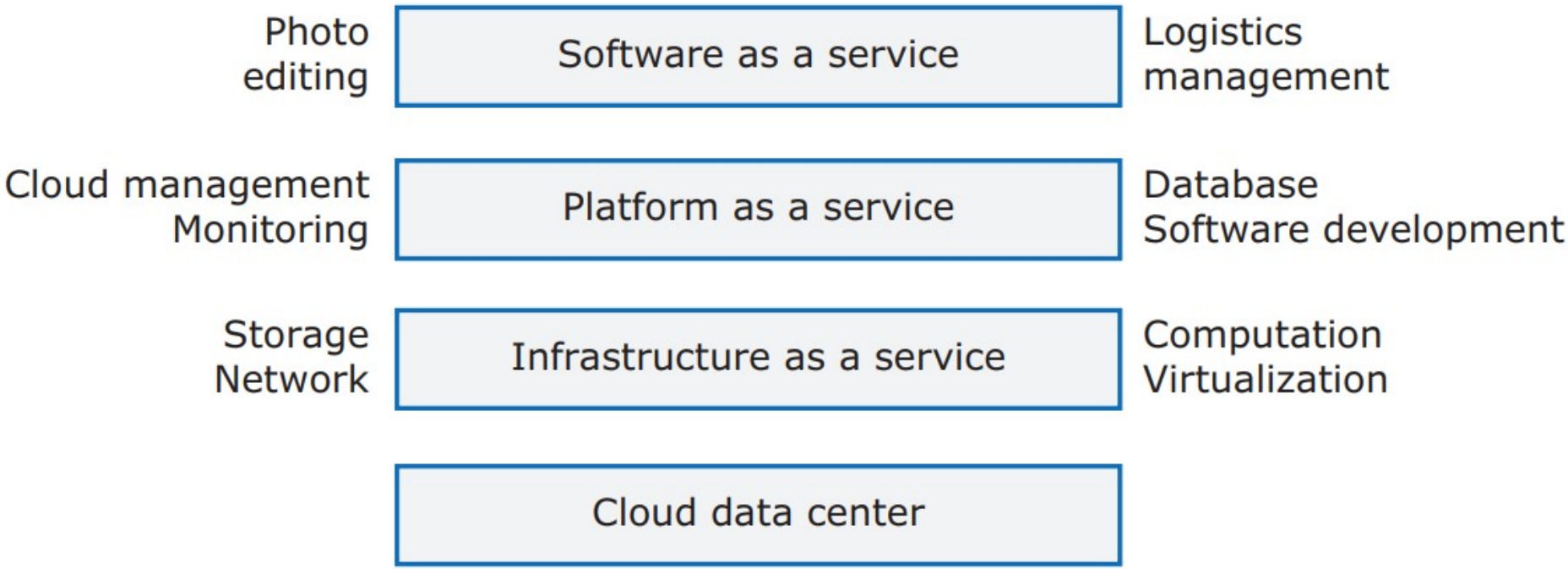

Figure 7: Everything as a service

Cloud-based Software

Infrastructure as a service (IaaS)

- This is a basic service level that all major cloud providers offer.
- IaaS provider provides the following services
  - **Compute:** Computing as a Service includes virtual central processing units and virtual main memory for the Vms that is provisioned to the end- users.
  - **Storage:** IaaS provider provides back-end storage for storing files.
  - **Network:** Network as a Service (NaaS) provides networking components such as routers, switches, and bridges for the Vms.
  - **Load balancers:** It provides load balancing capability at the infrastructure layer.
- The key benefits of using IaaS are
  - 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.
- Eg: facebook, Google App Engine, Google Colab, platforms for the data science.

Cloud-based Software

Software as a service (SaaS)

- SaaS is also known as "On-Demand Software".
- It is a software distribution model in which services are hosted by a cloud service provider.
- These services are available to end-users over the internet so, the end-users do not need to install any software on their devices to access these services.
- 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.

Cloud-based Software

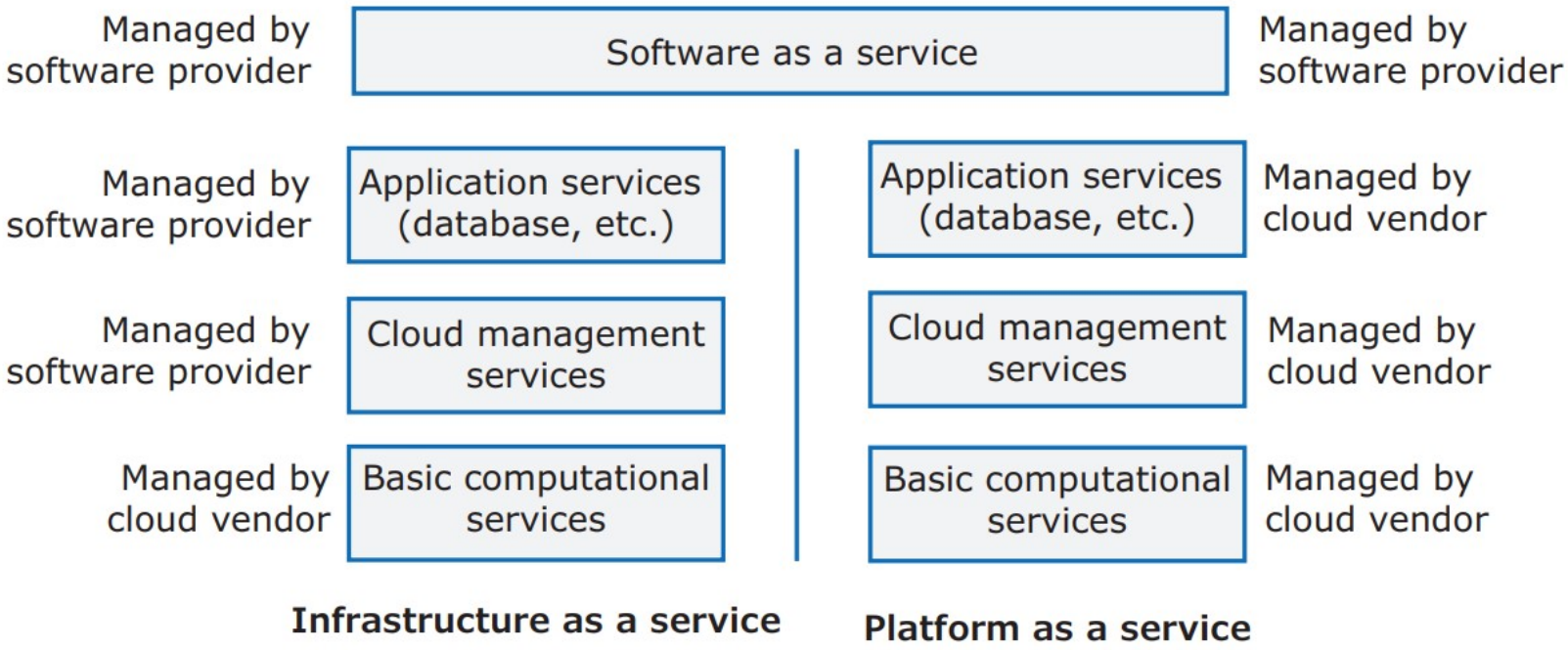

Figure 8: Management responsibilities for SaaS, IaaS, and PaaS

Microservices

- Microservices are small-scale, stateless services that have a single responsibility.
- Software products that use microservices are said to have a microservices architecture.
- If you need to create cloud-based software products that are adaptable, scalable, and resilient, then it is recommend to use a microservices architecture.

Microservices

- **Example of Microservices**
  - 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 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.
  - In principle, each of these features can be implemented as a separate service that uses a central shared database to hold authentication information.
    - A user interface service can then manage all aspects of user communication.
    - In a microservices architecture, however, these features are too large to be microservices.
    - To identify the microservices that might be used in the authentication system, you need to break down the coarse-grain features into more detailed functions.

Microservices

Figure shows what these functions might be for user registration and UID/password authentication.

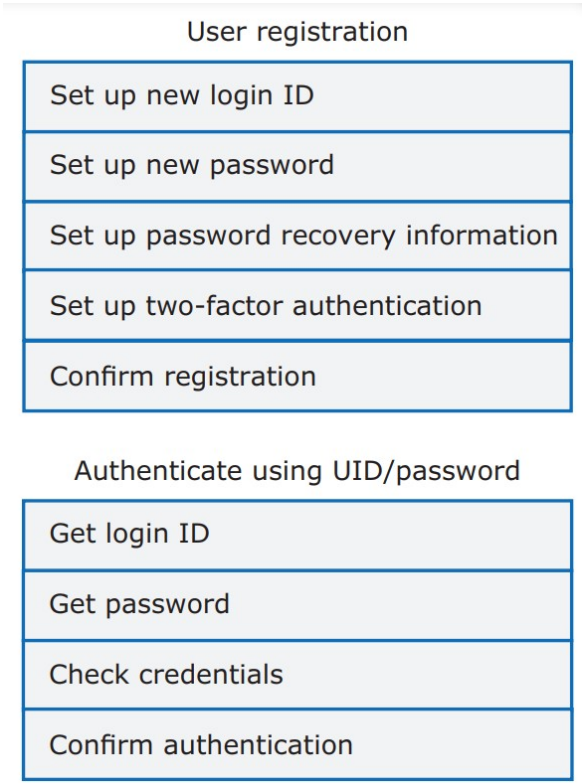

Figure 9: Functional breakdown of authentication features

Microservices

Figure shows the microservices that could be used to implement user authentication.

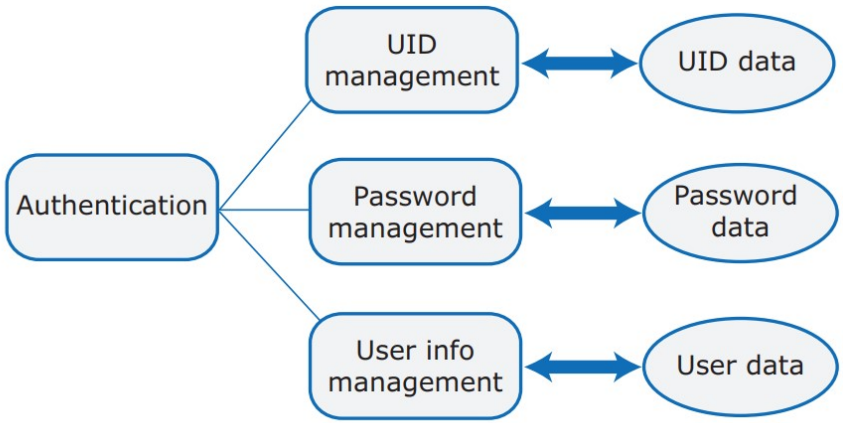

Figure 10: Authentication microservices

Microservices

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) 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 communication

- Microservices communicate by exchanging messages.
- A message that is sent between services includes some administrative information, a service request and the data required to deliver the requested service.
- Services return a response to service request messages.
  - An authentication service may send a message to a login service that includes the name input by the user.
  - The response may be a token associated with a valid user name or might be an error saying that there is no registered user.

Microservices architecture

- A microservices architecture is a tried and tested way of implementing a logical software architecture.
- This architectural style aims to address two fundamental problem with monolithic architecture
  1. The whole system has to be rebuilt, re-tested 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.
  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.

Microservices architecture

Benefits of microservices architecture

- Microservices are self-contained and run in separate processes.
- In cloud-based systems, each microservice may be deployed in its own container. This means a microservice can be stopped and restarted without affecting other parts of the system.
- If the demand on a service increases, service replicas can be quickly created and deployed. These do not require a more powerful server so 'scaling-out' is, typically, much cheaper than 'scaling up'.

Microservices architecture

A microservices architecture for a photo-printing system

- 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 chose 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.

Microservices architecture

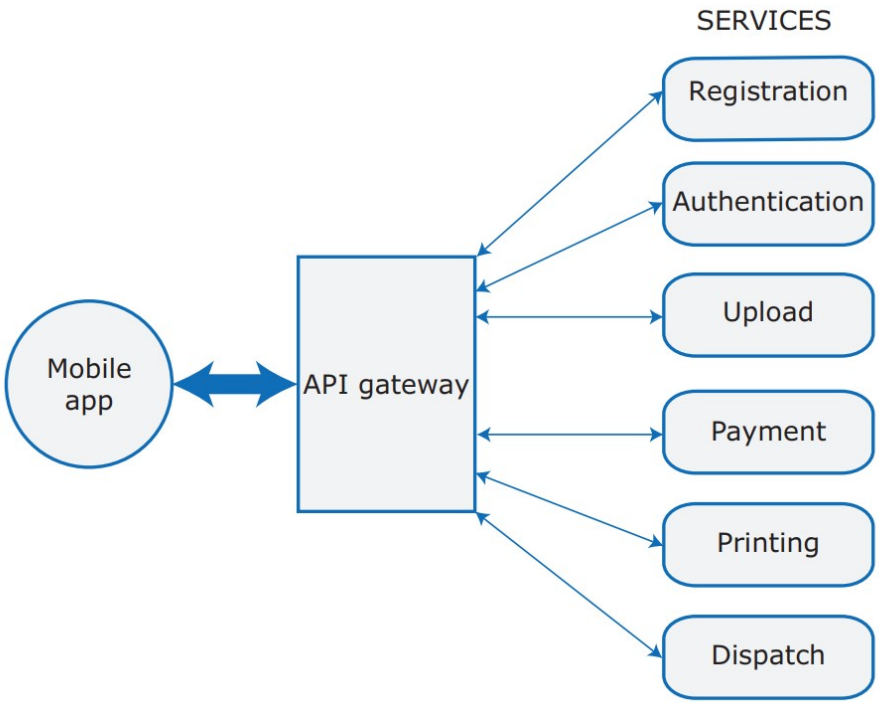

Figure 11: A microservices architecture for a photo-printing system

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 for microservices architecture

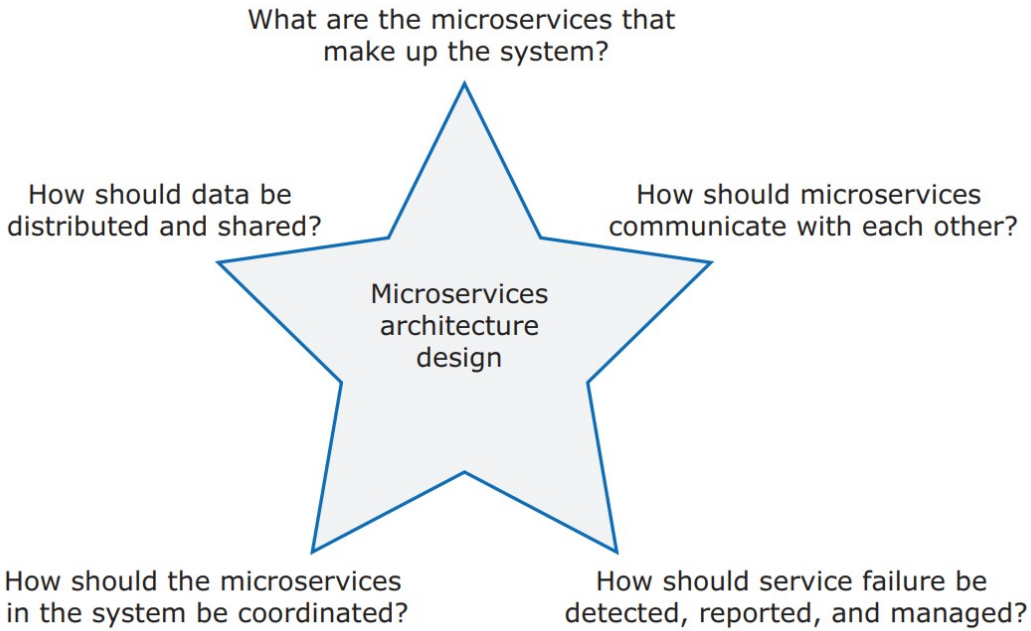

Figure 12: Key design questions for microservices architecture

Microservices architecture

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.
- 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.

Data Distributions and Sharing

- 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.

Failure management

- Three types of failures occur in microservice systems

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.

Figure 13: Failure types in a microservices system

The simplest way to report microservice failures is to use HTTP status codes, which indicate whether or not a request has succeeded.

Microservice deployment

- After a system has been developed and delivered, it has to be deployed on servers, monitored for problems, and updated as new versions become available.
- Traditionally, the tasks of managing an operational system were seen as separate from development.
- The system admin team had different skills from the system developers.
- When a system is composed of tens or even hundreds of microservices, deployment of the system is more complex than for monolithic systems.

Microservice deployment

- The service development teams decide which programming language, database, libraries, and other support software should be used to implement their service.
- Consequently, there is no "standard" deployment configuration for all services.
- Furthermore, services may change very quickly and there is the potential for a "deployment bottleneck" if a separate system admin team is faced with the problem of updating several services at the same time.

Microservice deployment

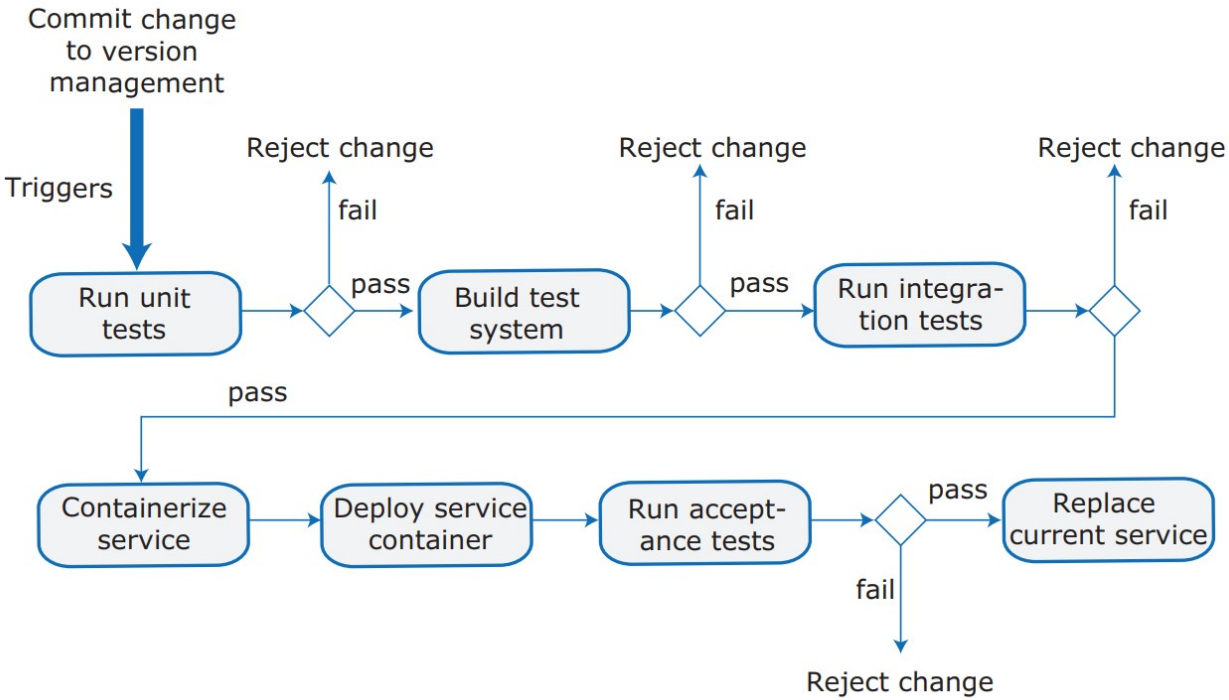

Figure 14: A continuous deployment pipeline