Engenharia de Software
(14341, 16230, 15386)

Software Architecture

**(adapted from *Software Engineering: International Version,* Ian Sommerville, Pearson, 2015, and
Engineering Software Products: An Introduction to Modern Software Engineering, Ian Sommerville, Pearson, 2020)**

Topics covered

✧ Software architecture

✧ Software architecture and components

Software architecture

◇ To create a reliable, secure and efficient product, you need to pay attention to architectural design which includes:

- its overall organization,
- how the software is decomposed into components,
- the server organization
- the technologies that you use to build the software. The architecture of a software product affects its performance, usability, security, reliability and maintainability.

Software architecture

♢ There are many different interpretations of the term 'software architecture'.

 ▪ Some focus on 'architecture' as a noun - the structure of a system and others consider 'architecture' to be a verb - the process of defining these structures.

The IEEE definition of software architecture

Software architecture

Architecture is the fundamental organization of a software system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.

Software architecture and components

◇ A component is an element that implements a coherent set of functionality or features.

◇ Software component can be considered as a collection of one or more services that may be used by other components.

◇ When designing software architecture, you don't have to decide how an architectural element or component is to be implemented.

◇ Rather, you design the component interface and leave the implementation of that interface to a later stage of the development process.

Access to services provided by software components

Services accessed through
the component API

Services accessed directly
by other components

API

| S1 | S2 | S3 |

| S4 | S5 | S6 |
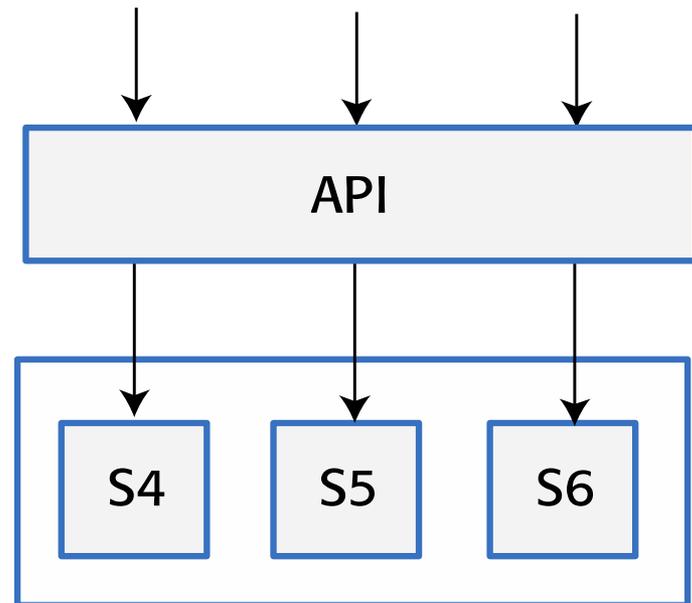
Component 1

Component 2

Why is architecture important? (1 of 3)

Why is architecture important?

✧ Architecture is important because the architecture of a system has a fundamental influence on the non-functional system properties.

✧ Architectural design involves understanding the issues that affect the architecture of your product and creating an architectural description that shows the critical components and their relationships.

Why is architecture important?

✧ Minimizing complexity should be an important goal for architectural designers.

   ▪ The more complex a system, the more difficult and expensive it is to understand and change.

   ▪ Programmers are more likely to make mistakes and introduce bugs and security vulnerabilities when they are modifying or extending a complex system.

Non-functional system quality attributes

Attribute	Key issue
Responsiveness	Does the system return results to users in a reasonable time?
Reliability	Do the system features behave as expected by both developers and users?
Availability	Can the system deliver its services when requested by users?
Security	Does the system protect itself and users' data from unauthorized attacks and intrusions?
Usability	Can system users access the features that they need and use them quickly and without errors?
Maintainability	Can the system be readily updated and new features added without undue costs?
Resilience	Can the system continue to deliver user services in the event of partial failure or external attack?

Centralized security architectures

✧ The benefits of a centralized security architecture are that it is easier to design and build protection and that the protected information can be accessed more efficiently.

✧ However, if your security is breached, you lose everything.

✧ If you distribute information, it takes longer to access all of the information and costs more to protect it.

✧ If security is breached in one location, you only lose the information that you have stored there.

Maintainability and performance

✧ This figure shows a system with two components (C1 and C2) that share a common database.

- Assume C1 runs slowly because it has to reorganize the information in the database before using it.

- The only way to make C1 faster might be to change the database. This means that C2 also has to be changed, which may, potentially, affect its response time.

```
        ┌─────────────────────────────┐
        │       User interface        │
        └─────────────────────────────┘
           ↕                    ↕
    ┌──────────┐          ┌──────────┐
    │    C1    │          │    C2    │
    └──────────┘          └──────────┘
           ↕                    ↕
        ┌─────────────────────────────┐
        │       Shared database       │
        └─────────────────────────────┘
```
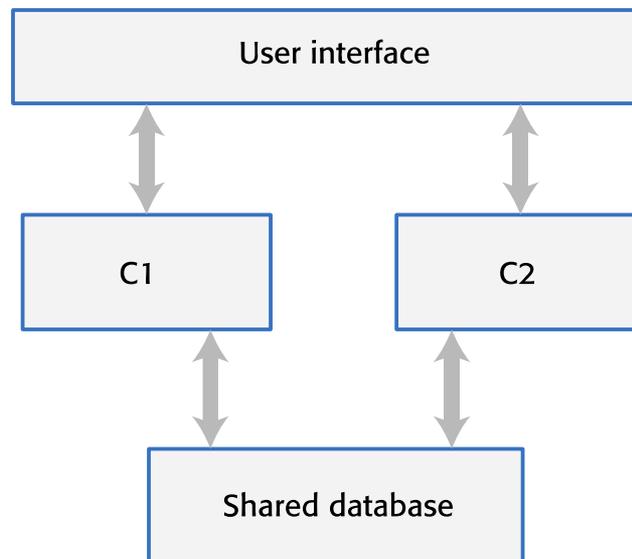
✧ This figure shows a different architecture is used where each component has its own copy of the parts of the database that it needs.

   ▪ If one component needs to change the database organization, this does not affect the other component.

User interface

C1		C2
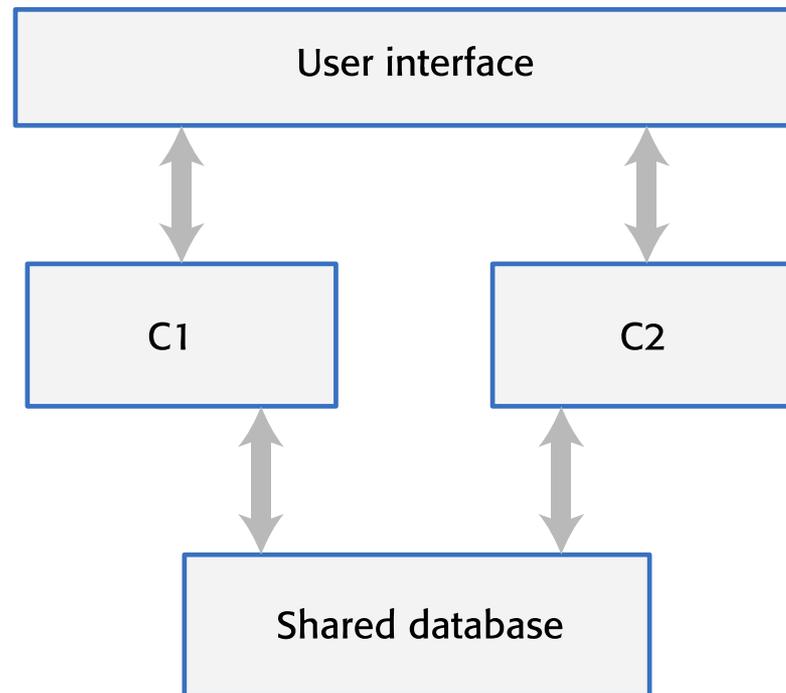
Shared database

Maintainability and performance (2 of 2)

✧ However, a multi-database architecture may run more slowly and may cost more to implement and change.

▪ A multi-database architecture needs a mechanism (component C3) to ensure that the data shared by C1 and C2 is kept consistent when it is changed.

Database reconciliation
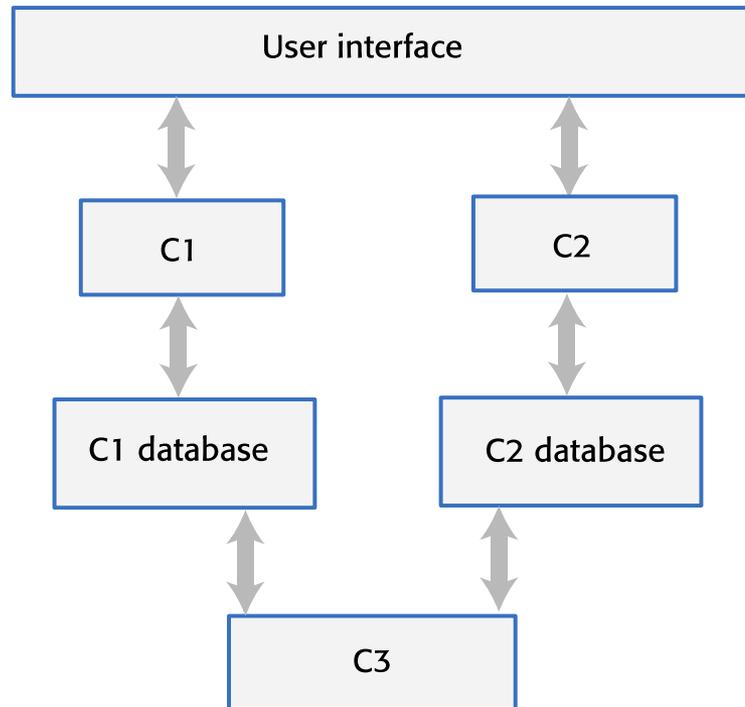
Issues that influence architectural decisions

Nonfunctional
product characteristics

Software
compatibility

Product
lifetime

Architectural
influences

Number of
users

Software
reuse

Issue	Architectural importance
Non-functional product characteristics	Non-functional product characteristics such as security and performance affect all users. If you get these wrong, your product is unlikely to be a commercial success. Unfortunately, some characteristics are opposing, so you can optimize only the most important.
Product lifetime	If you anticipate a long product lifetime, you need to create regular product revisions. You therefore need an architecture that can evolve, so that it can be adapted to accommodate new features and technology.
Software reuse	You can save a lot of time and effort if you can reuse large components from other products or open-source software. However, this constrains your architectural choices because you must fit your design around the software that is being reused.

The importance of architectural design issues

Issue	Architectural importance
Number of users	If you are developing consumer software delivered over the Internet, the number of users can change very quickly. This can lead to serious performance degradation unless you design your architecture so that your system can be quickly scaled up and down.
Software compatibility	For some products, it is important to maintain compatibility with other software so that users can adopt your product and use data prepared using a different system. This may limit architectural choices, such as the database software that you can use.

Trade off: Maintainability vs performance

✧ System maintainability is an attribute that reflects how difficult and expensive it is to make changes to a system after it has been released to customers.

- You improve maintainability by building a system from small self-contained parts, each of which can be replaced or enhanced if changes are required.

Trade off: Maintainability vs performance

⬧ In architectural terms, this means that the system should be decomposed into fine-grain components, each of which does one thing and one thing only.

  ▪ However, it takes time for components to communicate with each other. Consequently, if many components are involved in implementing a product feature, the software will be slower.

Trade off: Security vs usability

◇ You can achieve security by designing the system protection as a series of layers.

  ▪ An attacker has to penetrate all of those layers before the system is compromised.

◇ Layers might include system authentication layers, a separate critical feature authentication layer, an encryption layer and so on.

◇ Architecturally, you can implement each of these layers as separate components so that if one of these components is compromised by an attacker, then the other layers remain intact.

Authentication layers
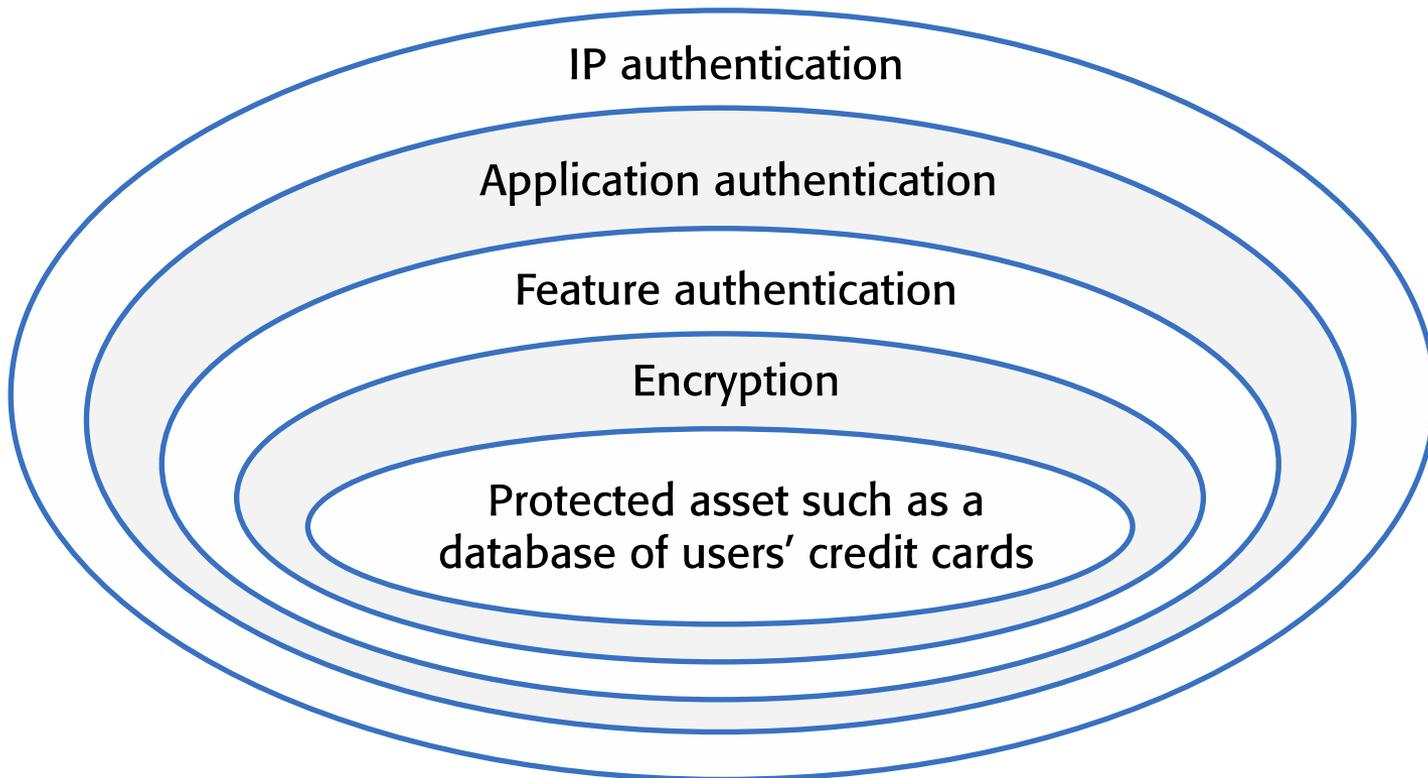
Usability issues

♢ A layered approach to security affects the usability of the software.

- Users have to remember information, like passwords, that is needed to penetrate a security layer. Their interaction with the system is inevitably slowed down by its security features.

- Many users find this irritating and often look for work-arounds so that they do not have to re-authenticate to access system features or data.

Usability issues (2 of 2)

✧ To avoid this, you need an architecture:

- that doesn't have too many security layers,
- that doesn't enforce unnecessary security,
- that provides helper components that reduce the load on users.

✧ Availability is particularly important in enterprise products, such as products for the finance industry, where 24/7 operation is expected.

✧ The availability of a system is a measure of the amount of 'uptime' of that system.

  ▪ Availability is normally expressed as a percentage of the time that a system is available to deliver user services.

Trade off: Availability vs time-to-market

✧ Architecturally, you achieve availability by having redundant components in a system.

   ▪ To make use of redundancy, you include sensor components that detect failure, and switching components that switch operation to a redundant component when a failure is detected.

✧ Implementing extra components takes time and increases the cost of system development. It adds complexity to the system and therefore increases the chances of introducing bugs and vulnerabilities.

Architectural design questions

✧ How should the system be organized as a set of architectural components, where each of these components provides a subset of the overall system functionality?

  ▪ The organization should deliver the system security, reliability and performance that you need.

✧ How should these architectural components be distributed and communicate with each other?

✧ What technologies should you use in building the system and what components should be reused?

Component organization

✧ Abstraction in software design means that you focus on the essential elements of a system or software component without concern for its details.

✧ At the architectural level, your concern should be on large-scale architectural components.

✧ Decomposition involves analysing these large-scale components and representing them as a set of finer-grain components.

✧ Layered models are often used to illustrate how a system is composed of components.

An architectural model of a document retrieval system

Web browser

User interaction	Local input validation	Local printing

User interface management

Authentication and authorization	Form and query manager	Web page generation

Information retrieval

Search	Document retrieval	Rights management	Payments	Accounting

Document index

Index management	Index querying	Index creation
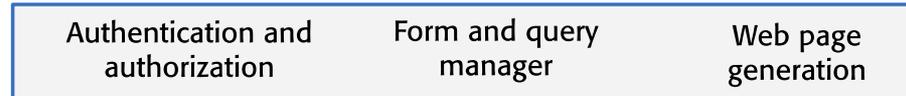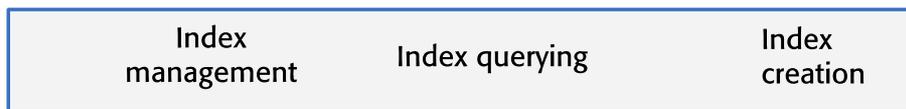
Basic services

Database query	Query validation	Logging	User account management

Databases

| DB1	DB2	DB3	DB4	DB5
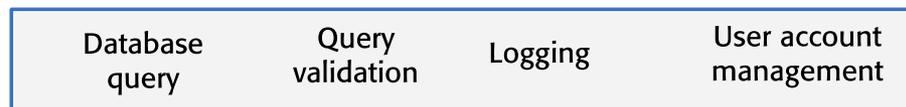
Architectural complexity

✧ Complexity in a system architecture arises because of the number and the nature of the relationships between components in that system.

✧ When decomposing a system into components, you should try to avoid unnecessary software complexity.

■ *Localize relationships:* If there are relationships between components A and B, these are easier to understand if A and B are defined in the same module.

Architectural complexity (2 of 2)

 ² When decomposing a system into components, you should try to avoid unnecessary software complexity.

   ▪ *Reduce shared dependencies:* Where components A and B depend on some other component or data, complexity increases because changes to the shared component mean you have to understand how these changes affect both A and B.

 ² It is always preferable to use local data wherever possible and to avoid sharing data if you can.

Examples of component relationships

C1 is-part-of C2

C2   C1

C1 uses C2

C1
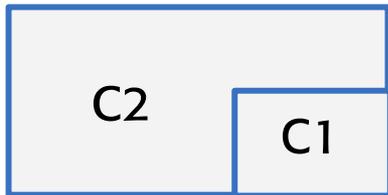
calls

C2
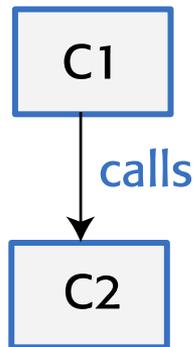
C1 is-located-with C2

C1   C2
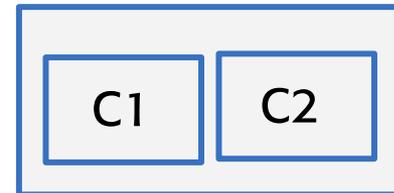
C1 shares-data-with C2

C1 ⟷ data ⟷ C2

Architectural design guidelines

Separation of concerns
Organize your architecture
into components that focus on
a single concern
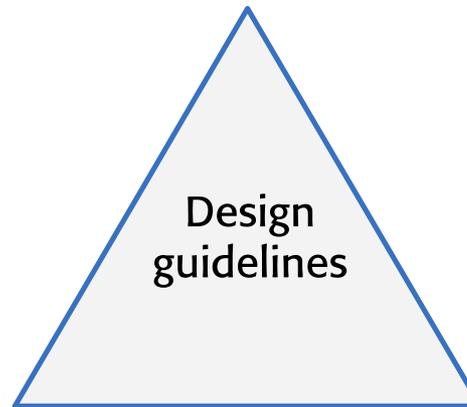
Design
guidelines

Stable interfaces
Design component
interfaces that are coherent
and that change slowly

Implement once
Avoid duplicating
functionality at different
places in your architecture

Design guidelines and layered architectures

- ✧ Each layer is an area of concern and is considered separately from other layers.
  - ▪ The top layer is concerned with user interaction, the next layer down with user interface management, the third layer with information retrieval and so on.
- ✧ Within each layer, the components are independent and do not overlap in functionality.
  - ▪ The lower layers include components that provide general functionality so there is no need to replicate this in the components in a higher level.

Design guidelines and layered architectures (2 of 2)

◇ The architectural model is a high-level model that does not include implementation information.

- Ideally, components at level X (say) should only interact with the APIs of the components in level X-1. That is, interactions should be between layers and not across layers.

Cross-cutting concerns

✧ Cross-cutting concerns are concerns that are systemic, that is, they affect the whole system.

✧ In a layered architecture, cross-cutting concerns affect all layers in the system as well as the way in which people use the system.

✧ Cross-cutting concerns are completely different from the functional concerns represented by layers in a software architecture.

✧ Every layer has to take them into account and there are inevitably interactions between the layers because of these concerns.

✧ The existence of cross-cutting concerns is the reason why modifying a system after it has been designed to improve its security is often difficult.

Cross-cutting concerns

Security   Performance   Reliability

User interface

Application

Infrastructure

Operating system

Hardware
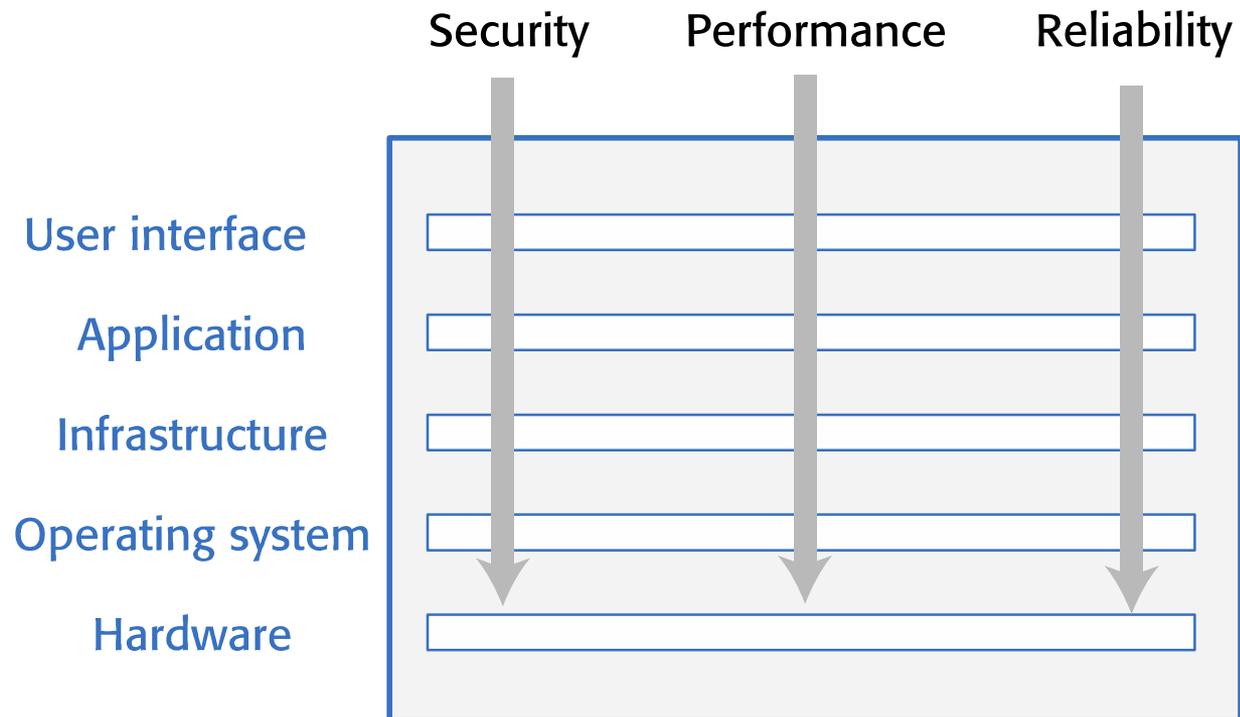
Security as a cross-cutting concern

Security architecture

Different technologies are used in different layers, such as an SQL database or a Firefox browser. Attackers can try to use vulnerabilities in these technologies to gain access. Consequently, you need protection from attacks at each layer as well as protection at lower layers in the system from successful attacks that have occurred at higher-level layers.

If there is only a single security component in a system, this represents a critical system vulnerability. If all security checking goes through that component and it stops working properly or is compromised in an attack, then you have no reliable security in your system. By distributing security across the layers, your system is more resilient to attacks and software failure (remember the *Rogue* One example earlier in the chapter).

A generic layered architecture for a web-based application

Browser-based or mobile user interface

Authentication and user interaction management

Application-specific functionality

Basic shared services

Transaction and database management

Layer functionality in a web-based application

Layer	Explanation
Browser-based or mobile user Interface	A web browser system interface in which HTML forms are often used to collect user input. Javascript components for local actions, such as input validation, should also be included at this level. Alternatively, a mobile interface may be implemented as an app.
Authentication and UI management	A user interface management layer that may include components for user authentication and web page generation.
Application-specific functionality	An "application" layer that provides functionality of the application. Sometimes this may be expanded into more than one layer.

Layer functionality in a web-based application

Layer	Explanation
Basic shared services	A shared services layer that includes components that provide services used by the application layer components.
Database and transaction Management	A database layer that provides services such as transaction management and recovery. If your application does not use a database, then this may not be required.

iLearn architectural design principles

Principle	Explanation
Replaceability	It should be possible for users to replace applications in the system with alternatives and to add new applications. Consequently, the list of applications included should not be hardwired into the system.
Extensibility	It should be possible for users or system administrators to create their own versions of the system, which may extend or limit the "standard" system.
Age-appropriate	Alternative user interfaces should be supported so that ageappropriate interfaces for students at different levels can be created.
Programmability	It should be easy for users to create their own applications by linking existing applications in the system.
Minimum work	Users who do not wish to change the system should not have to do extra work so that other users can make changes.

iLearn design principles

♢ Our goal in designing the iLearn system was to create an adaptable, universal system that could be easily updated as new learning tools became available.

- This means that it must be possible to change and replace components and services in the system (principles (1) and (2)).

- Because the potential system users spanned an age range from 3 to 18, we needed to provide age-appropriate user interfaces and to make it easy to choose an interface (principle (3)).

- Principle (4) also contributes to system adaptability and principle (5) was included to ensure that this adaptability did not adversely affect users who did not require it.

✧ These principles led us to an architectural design decision that the iLearn system should be service-oriented.

✧ Every component in the system is a service. Any service is potentially replaceable and new services can be created by combining existing services. Different services delivering comparable functionality can be provided for students of different ages.

✧ Service integration

- *Full integration:* Services are aware of and can communicate with other services through their APIs.

- *Partial integration:* Services may share service components and databases but are not aware of and cannot communicate directly with other application services.

- *Independent:* These services do not use any shared system services or databases and they are unaware of any other services in the system. They can be replaced by any other comparable service.

A layered architectural model of the iLearn system

User interface

Web browser	iLearn app

User interface management

Interface creation	Forms management	Interface delivery	Login

Configuration services

Group configuration	Application configuration	Security configuration	User interface configuration	Setup service

Application services

Archive access   Word processor   Video conf.   Email and messaging   User installed applications

Blog   Wiki   Spreadsheet   Presentation   Drawing

Integrated services

Resource discovery   User analytics   Virtual learning environment   Authentication and authorization
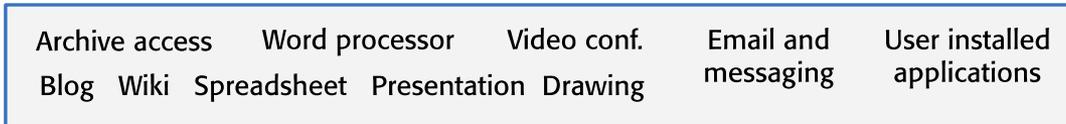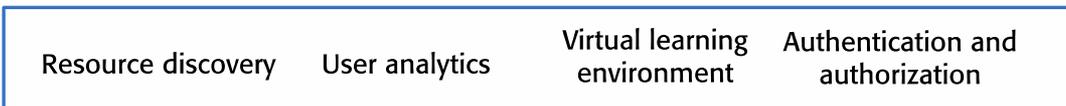
Shared infrastructure services

Authentication   Logging and monitoring   Application interfacing

User storage   Application storage   Search
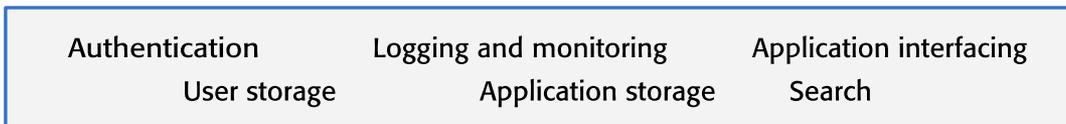
Distribution architecture

✧ The distribution architecture of a software system defines the servers in the system and the allocation of components to these servers.

✧ Client-server architectures are a type of distribution architecture that is suited to applications where clients access a shared database and business logic operations on that data.

✧ In this architecture, the user interface is implemented on the user's own computer or mobile device.

  ▪ Functionality is distributed between the client and one or more server computers.

Client–server architecture
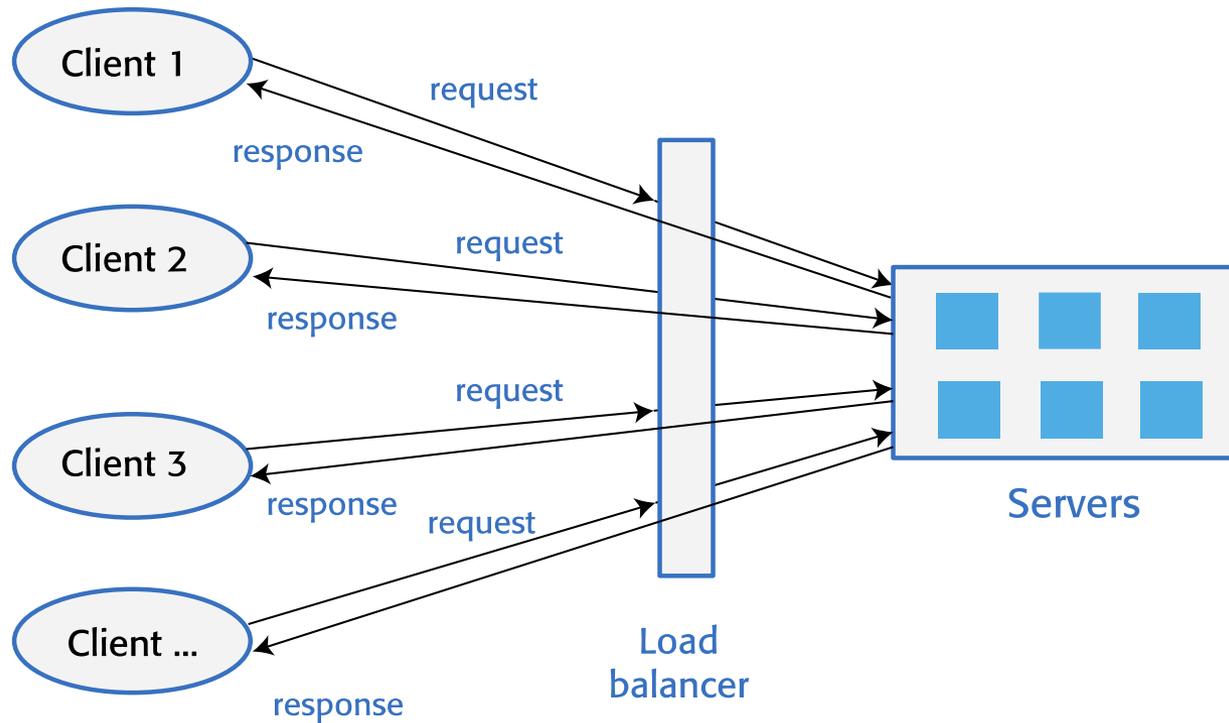
Client–server architecture

Name	Client-server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.

The Model-View-Controller (MVC) pattern

CLIENT

Browser

User inputs

Page to display

User changes

Controller

View

View update request

Change notification

View refresh request

Model

SERVER

The Model-View-Controller (MVC) pattern

| Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into **three logical components** that interact with each other. The **Model** component manages the system data and associated operations on that data. The **View** component defines and manages how the data is presented to the user. The **Controller** component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.

Client-server communication (1 of 2)

✧ Client-server communication normally uses the HTTP protocol.

  ▪ The client sends a message to the server that includes an instruction such as GET or POST along with the identifier of a resource (usually a URL) on which that instruction should operate. The message may also include additional information, such as information collected from a form.

Client-server communication

♢ HTTP is a text-only protocol so structured data has to be represented as text. There are two ways of representing this data that are widely used, namely XML and JSON.

  ▪ XML is a markup language with tags used to identify each data item.

 ▪ JSON is a simpler representation based on the representation of objects in the Javascript language.

Multi-tier client–server architecture
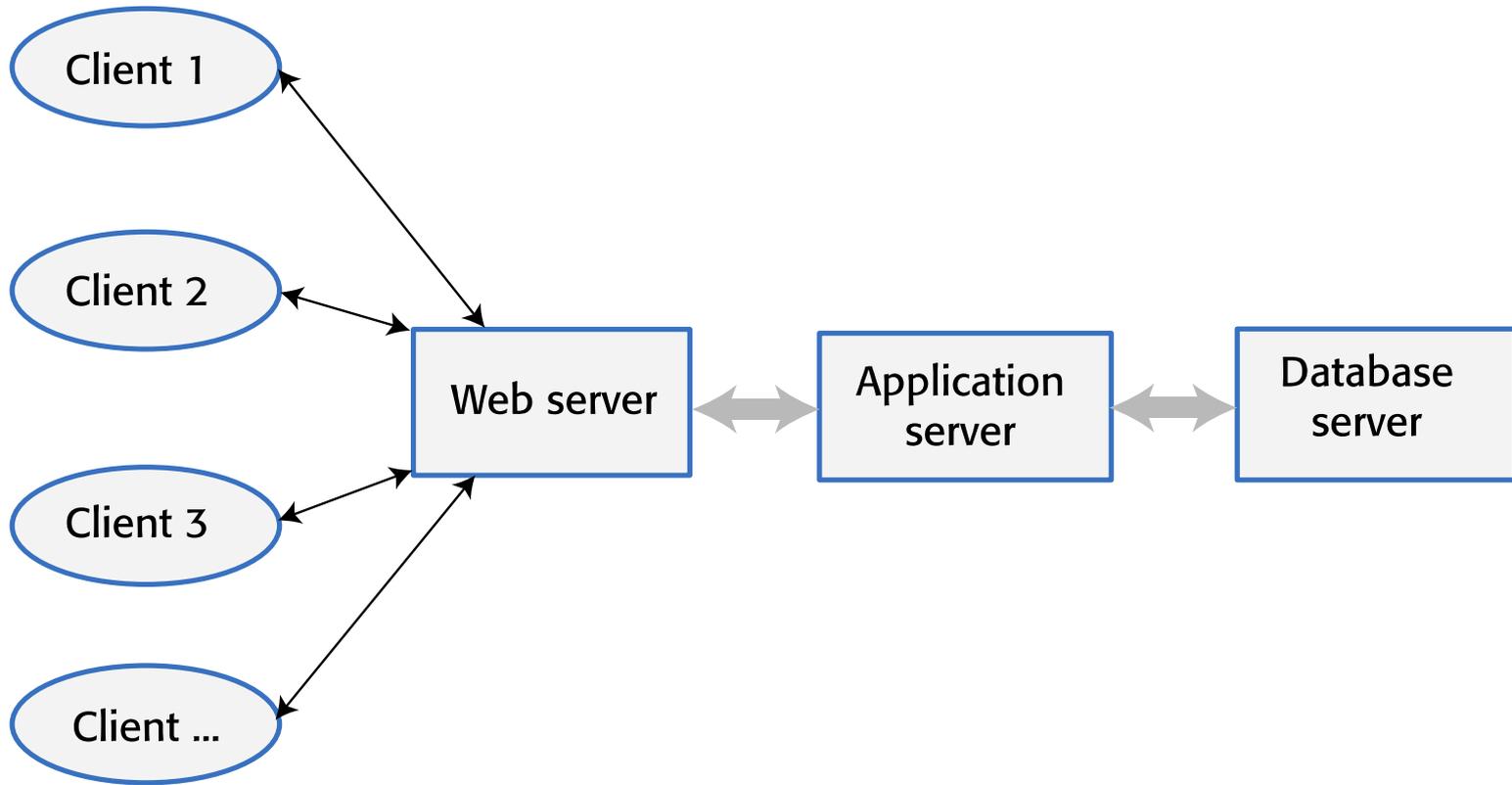
Service-oriented architecture

♢ Services in a service-oriented architecture are stateless components, which means that they can be replicated and can migrate from one computer to another.

♢ Many servers may be involved in providing services

♢ A service-oriented architecture is usually easier to scale as demand increases and is resilient to failure.

Service-oriented architecture
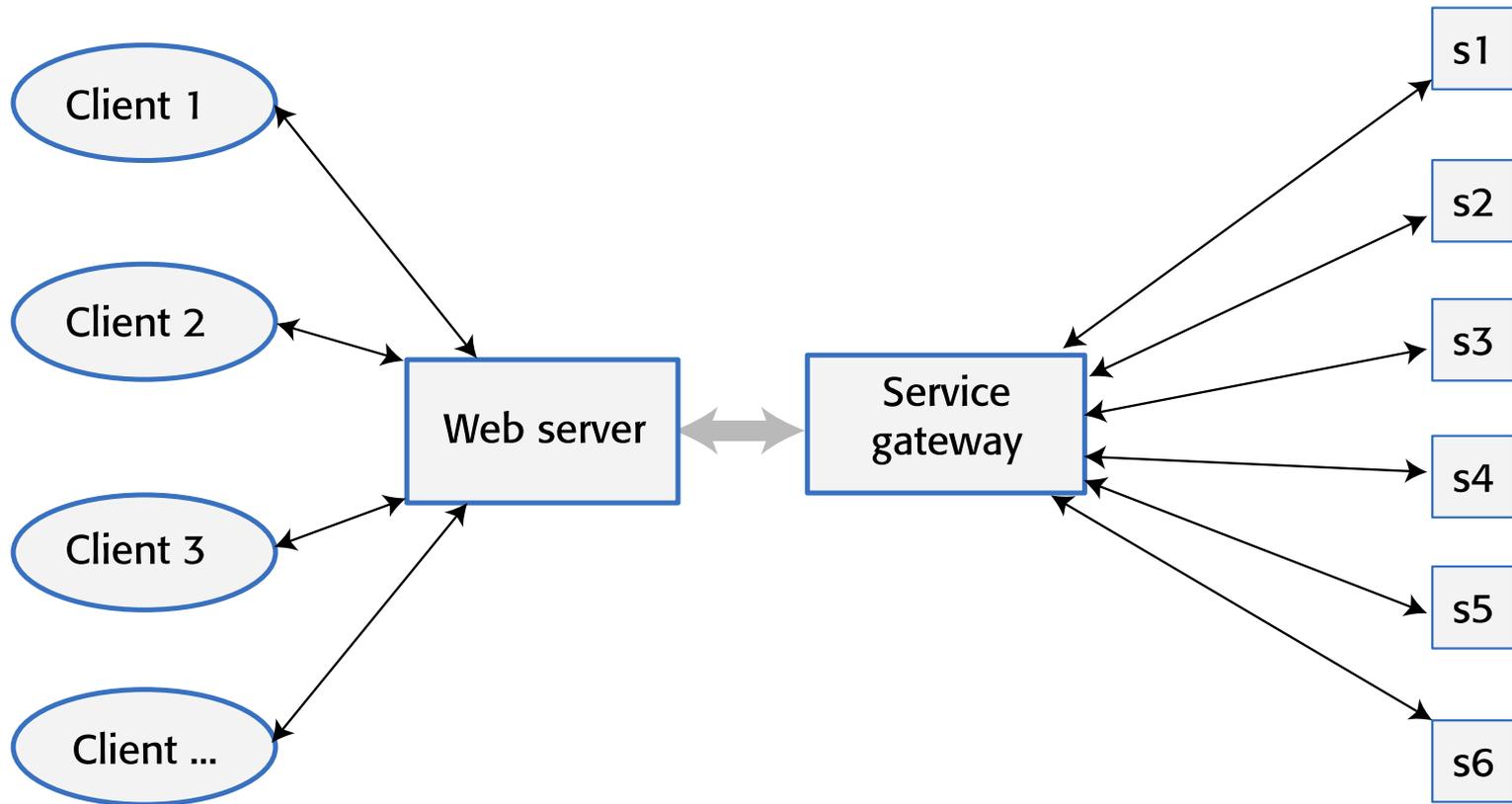
Issues in architectural choice (1 of 2)

✧ Data type and data updates

- If you are mostly using structured data that may be updated by different system features, it is usually best to have a single shared database that provides locking and transaction management. If data is distributed across services, you need a way to keep it consistent and this adds overhead to your system.

✧ Change frequency

- If you anticipate that system components will be regularly changed or replaced, then isolating these components as separate services simplifies those changes

Issues in architectural choice (2 of 2)

 ⬦ The system execution platform

 - If you plan to run your system on the cloud with users accessing it over the Internet, it is usually best to implement it as a service-oriented architecture because scaling the system is simpler.

 - If your product is a business system that runs on local servers, a multi-tier architecture may be more appropriate.

Technology choices

Technology	Design decision
Database	Should you use a relational SQL database or an unstructured NoSQL database?
Platform	Should you deliver your product on a mobile app and/or a web platform?
Server	Should you use dedicated in-house servers or design your system to run on a public cloud? If a public cloud, should you use Amazon, Google, Microsoft, or some other option?
Open source	Are there suitable open-source components that you could incorporate into your products?
Development tools	Do your development tools embed architectural assumptions about the software being developed that limit your architectural choices?

Database (1 of 2)

♢ There are two kinds of database that are now commonly used:

- Relational databases, where the data is organised into structured tables.
- NoSQL databases, in which the data has a more flexible, user-defined organization.

♢ Relational databases, such as MySQL, are particularly suitable for situations where you need transaction management, and the data structures are predictable and fairly simple.

Database (2 of 2)

✧ NoSQL databases, such as MongoDB, are more flexible and potentially more efficient than relational databases for data analysis.

- NoSQL databases allow data to be organized hierarchically rather than as flat tables and this allows for more efficient concurrent processing of 'big data'.

Delivery platform (1 of 3)

◇ Delivery can be as a web-based or a mobile product or both

◇ Mobile issues:

   ▪ *Intermittent connectivity* You must be able to provide a limited service without network connectivity.

   ▪ *Processor power* Mobile devices have less powerful processors, so you need to minimize computationally-intensive operations.

Delivery platform (2 of 3)

♢ Mobile issues:

- *Power management* Mobile battery life is limited so you should try to minimize the power used by your application.

- *On-screen keyboard* On-screen keyboards are slow and error-prone. You should minimize input using the screen keyboard to reduce user frustration.

Delivery platform (3 of 3)

◇ To deal with these differences, you usually need separate browser-based and mobile versions of your product front-end.

- You may need a completely different decomposition architecture in these different versions to ensure that performance and other characteristics are maintained.

Server

Server (1 of 2)

✧ A key decision that you have to make is whether to design your system to run on customer servers or to run on the cloud.

✧ For consumer products that are not simply mobile apps it almost always makes sense to develop for the cloud.

Server

♢ For business products, it is a more difficult decision.

  ▪ Some businesses are concerned about cloud security and prefer to run their systems on in-house servers. They may have a predictable pattern of system usage so there is less need to design your system to cope with large changes in demand.

♢ An important choice you have to make if you are running your software on the cloud is which cloud provider to use.

✧ Open source software is software that is available freely, which you can change and modify as you wish.

■ The advantage is that you can reuse rather than implement new software, which reduces development costs and time to market.

■ The disadvantages of using open-source software is that you are constrained by that software and have no control over its evolution.

Open source (2 of 2)

◇ The decision on the use of open-source software also depends on the availability, maturity and continuing support of open source components.

◇ Open source license issues may impose constraints on how you use the software.

◇ Your choice of open source software should depend on the type of product that you are developing, your target market and the expertise of your development team.

Development tools (1 of 2)

♢ Development technologies, such as a mobile development toolkit or a web application framework, influence the architecture of your software.

- These technologies have built-in assumptions about system architectures and you have to conform to these assumptions to use the development system.

Development tools

² The development technology that you use may also have an indirect influence on the system architecture.

- Developers usually favour architectural choices that use familiar technologies that they understand. For example, if your team have a lot of experience of relational databases, they may argue for this instead of a NoSQL database.

✧ Software architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

✧ The architecture of a software system has a significant influence on non-functional system properties such as reliability, efficiency and security.

✧ Architectural design involves understanding the issues that are critical for your product and creating system descriptions that shows components and their relationships.

✧ The principal role of architectural descriptions is to provide a basis for the development team to discuss the system organization. Informal architectural diagrams are effective in architectural description because they are fast and easy to draw and share.

✧ System decomposition involves analyzing architectural components and representing them as a set of finer-grain components.

Key points (3 of 4)

♢ To minimize complexity, you should separate concerns, avoid functional duplication and focus on component interfaces.

♢ Web-based systems often have a common layered structure including user interface layers, application-specific layers and a database layer.

♢ The distribution architecture in a system defines the organization of the servers in that system and the allocation of components to these servers.

Key points (4 of 4)

⬦ Multi-tier client-server and service-oriented architectures are the most commonly used architectures for web-based systems.

⬦ Making decisions on technologies such as database and cloud technologies are an important part of the architectural design process.