Software Engineering – Lecture 5
Architectural Design

Topics covered

Software architecture

Architectural design

Architectural styles

Architectures for software products

Software design

Def. **Design** = the process of transforming the requirements in *design*.

Architectural design – developing an abstract high-level model of the system.

- An early stage of the system design process.
- Represents the link between *specification* process and *design* process.
- Often carried out in parallel with some specification activities.
- Involves *identifying major system components* and their *communications*.

Detailed design – decomposing and refining the components of the architecture.

Software architecture

Def. A ***software architecture*** for a system is the set of *structures* needed to reason about the system, which comprise software *elements*, *relations* among them, and external visible *properties* of both.

(Bass, Clements, and Kazman)

Def. 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.

(IEEE definition of software architecture)

'architecture' as a *noun* - the *structure* of a system

'architecture' to be a *verb* - the *process* of defining these structures.

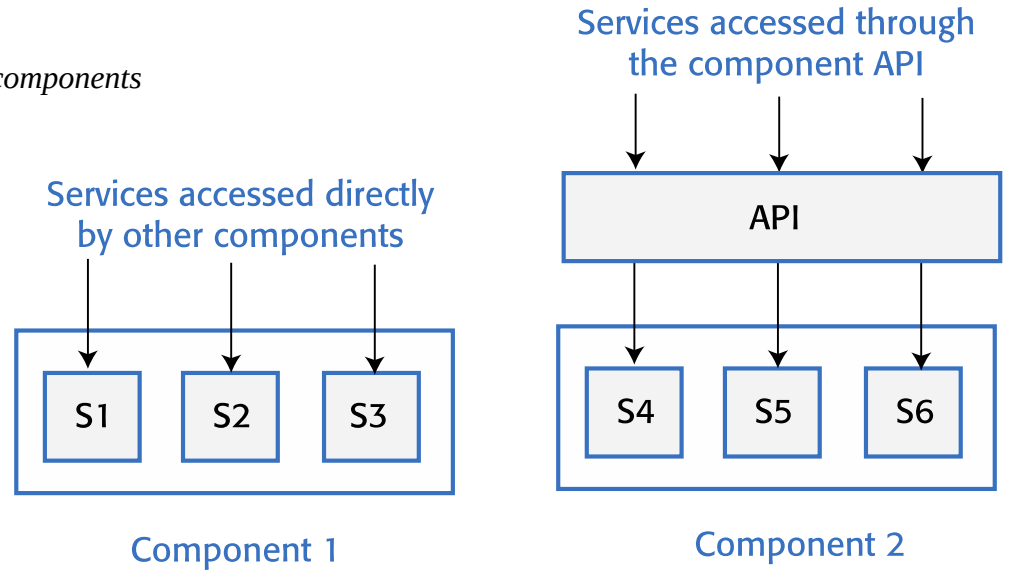
Software architecture and components

Component = element that implements a coherent set of functionality or features.

Provides a collection of one or more services that may be used by other components.

Designing software architecture ⇒ design the component interface (and leave the implementation of that interface to a later stage of the development process).

Figure 4.1 Access to services provided by software components
Ian Sommerville – Engineering Software Products



Advantages of explicit architecture

Stakeholder communication

- Architecture may be used as a focus of discussion by system stakeholders.

System analysis

- Allows analyses of whether the system can meet its extra-functional requirements.

Large-scale reuse

- The architecture may be reusable across a range of systems.
- Product-line architectures may be developed.

Architecture and quality attributes

Any software system has a software architecture (even it has not been deliberately designed).

The simplest system is made of one element in relation to itself.

Software architecture fulfills functional requirements an also promotes quality attributes.

Software architecture is critical in obtaining the quality attributes.

Architecture design is driven mainly by the requirements for quality attributes.

Quality attributes : examples

Responsiveness - The system return results to users in a reasonable time.

Reliability – The system features behave as expected by both developers and users.

Availability - The system delivers its services when requested by users.

Security - The system protect itself and users' data from unauthorized attacks and intrusions.

Usability - System users can access the features that they need and use them quickly and without errors.

Maintainability - The system can be readily updated and new features added without undue costs.

Resilience - The system can continue to deliver user services in the event of partial failure or external attack.

Architectural solutions for quality attributes

Security

Use a layered architecture with critical assets in the inner layers.

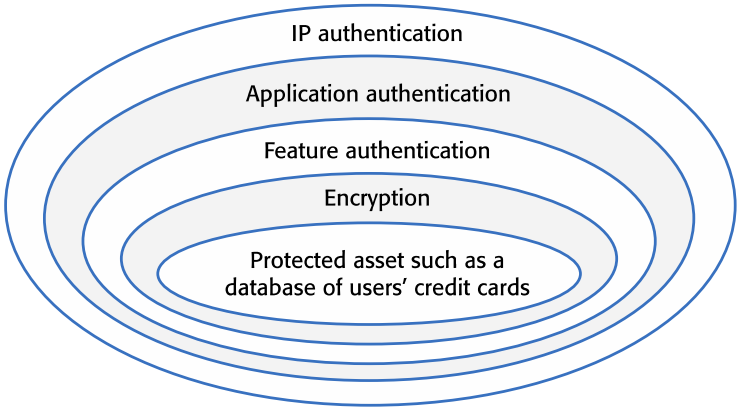


Figure 4.5 Authentication layers
Ian Sommerville – Engineering Software Products

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.

Each layer is a separate component so that if one of these components is compromised by an attacker, then the other layers remain intact.

Architectural solutions for quality attributes

Security discussion :

Security architecture with centralized information:

- easier to design and build protection
- protected information can be accessed more efficiently.

security breached $\Rightarrow$ everything is lost.

Security architecture with distributed information:

- costs more to protect it
- takes longer to access all of the information

security breached $\Rightarrow$ only the information stored in one location is lost.

Architectural solutions for quality attributes

Safety

Isolate safety-critical features in a small number of sub-systems.

Availability

Include redundant components and mechanisms for fault tolerance.

Performance

Localize critical operations and minimize communications. Implies use of large-grain components (rather than fine-grain ones).

Maintainability

Use fine-grain, replaceable components.

Architectural conflicts

Architectural conflict appears when two ***quality attributes*** are in tension, meaning ***increasing one results in decreasing the other***.

Examples.

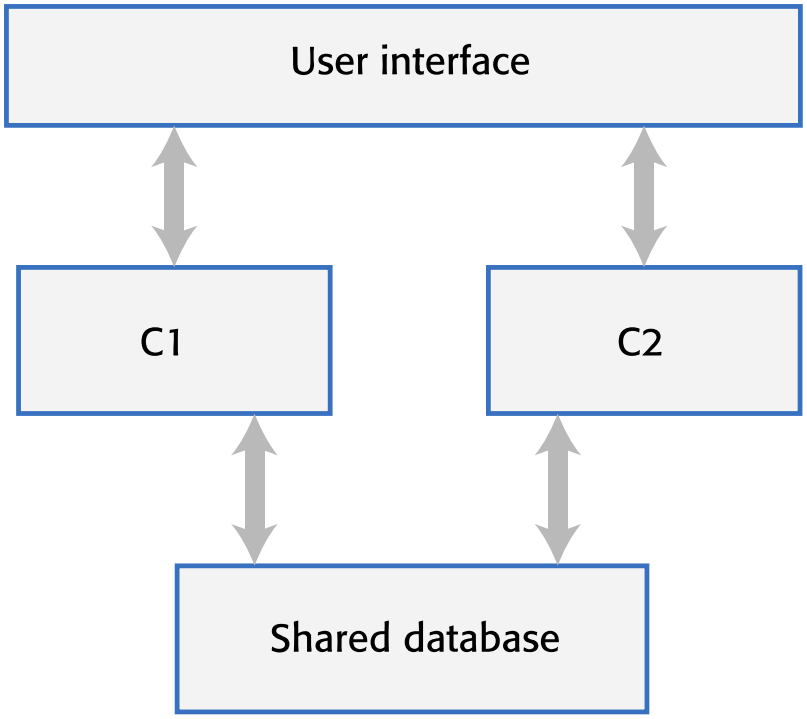
- Using large-grain components improves performance but reduces maintainability.
- Introducing redundant data improves availability but makes security more difficult.
- Localising safety-related features usually means more communication, resulting into degraded performance.

⇓

Architectural design implies a series of ***decisions*** in order to realize the best ***trade-off*** in fulfilling ***quality requirements***.

Maintainability and performance

Figure 4.2 Shared database architecture
Ian Sommerville – Engineering Software Products



Example v.1:

- 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.

⇓

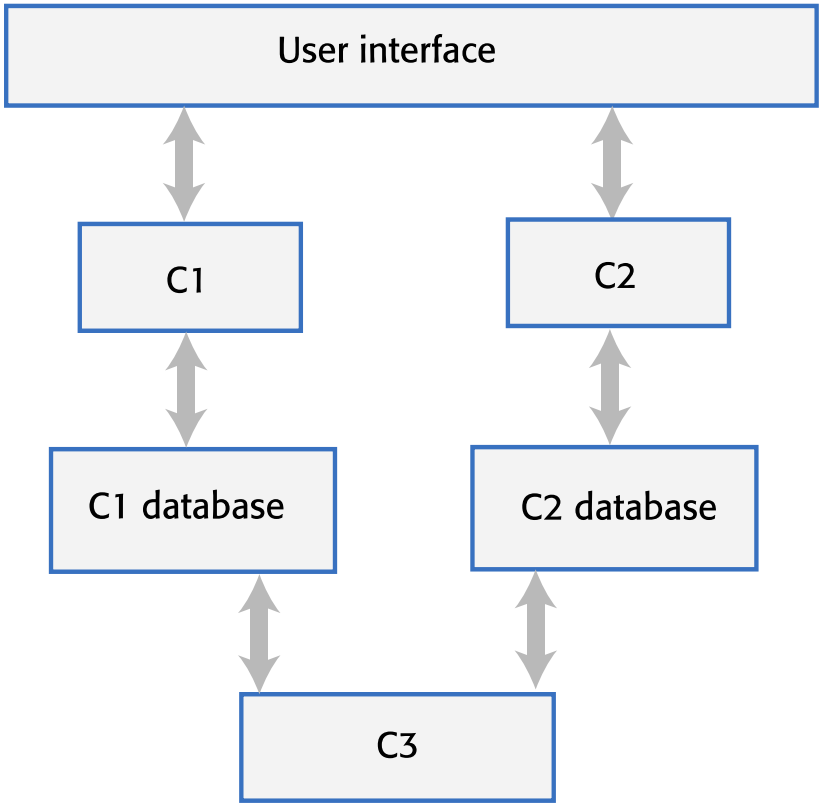
- C2 also has to be changed

However

- Its response time may, potentially, be affected.

Maintainability and performance

Figure 4.3 Multiple database architecture
Ian Sommerville – Engineering Software Products



Example v.2:

- 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.

BUT

- Component C3 is needed to ensure that the data shared by C1 and C2 is kept consistent when it is changed.

However

- A multi-database architecture may run more slowly and may cost more to implement and change.

Trade off: Maintainability vs performance

Maintainability = *difficulty* and the *cost* of changes after system release.

Performance = response time at service request

Architectural solution :
-Decomposition into fine-grain, replaceable components.

Consequence :
-Inter-component communication time.

⇒

Consequence :
-Slow response ⇒
-Reduced performance

Trade-off

Example: Number of components

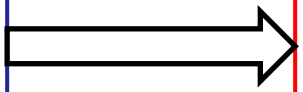
Trade off: Security vs usability

Security = system ability to resist to unauthorized attempts to use data or services, while offering access to legitimate users.

Usability = easy to learn and easy to efficiently access and use the software system.

Architectural solution
-Layered architecture with critical assets in the inner layers.

Consequences :
-Information (ex. passwords) needed to penetrate each security layer.
-Slowed interaction with the system



Consequences :
Effort to remember
User dissatisfaction

Trade-off

Example : Number of security layers.

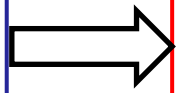
Trade off: Availability vs market competitivity

Availability = measure (percentage of the total time) of the amount of 'uptime' of a system.

Market competitivity = reduced development time and cost.

Architectural solution
-Redundant components

Consequences :
-Sensor components : detect failure,
-Switching components : switch operation to a redundant component



Consequences :
-Time to implement extra-components
-Increased complexity
-Risks to introduce bugs and vulnerabilities

Trade-off

Example: Redundancy degree

.

Formative evaluation

1. Realize the correct mapping between the quality attribute and the solution for improving that quality.

2. What is an architectural conflict and which is the method to solve it ?

https://forms.gle/e3WBcVMSTDpeCafH7

Topics covered

Software architecture

Architectural design

Architectural styles

Architectures for software products

Software architecture

Def. ***Architectural design*** is the design process for identifying

- The *sub-systems* making up a system
- The *framework for sub-system control and communication.*

The output of this design process is a ***description of the software architecture.***

Software system - organized set of architectural components.

Architectural design implies to define :

- component functionality (a subset of the overall system functionality)
- components distribution and intercommunication
- technologies used
- reused components

Architectural design decisions

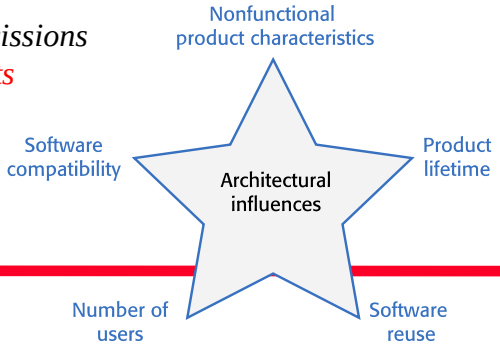
Architectural design – creative process, depending on the type of system being developed.

BUT: A number of common decisions span all design processes and these decisions affect extra-functional characteristics of the system.

Example:

- Is there a *generic application architecture* that can be used?
- How will the system be *decomposed* into modules?
- What *approach* will be used *to structure* the system?
- What *architectural styles* are appropriate?

Figure 4.4 Issues that influence architectural decissions
Ian Sommerville – Engineering Software Products



Influences on architectural design decisions

Quality attributes of the product
Define the overall quality of the software product perceived on the market. Some qualities are opposing, so only the most important will be optimized.

Product lifetime
A long product lifetime ⇒ regular product revisions ⇒ evolvable architecture, able to be adapted to accommodate new features and technology.

Software reuse
Saving time and effort by reusing large components from other products or open-source software ⇒ architectural choices constrained to fit the design around the software being reused.

Number of users
Quickly variable number of users ⇒ serious performance degradation ⇒ scalable (up and down) architecture.

Software compatibility
Compatibility with existing software and data ⇒ limited architectural choices.

Architectural design : steps

1. Defining the relationships (interfaces) of the system to its *context* made of human users, other systems and devices.

Iterative:

2. Decomposing the system in elements (sub-systems, modules, components) that interact in order to fulfill functional and quality requirements.

3. Establishing *relationships* (interfaces) among elements.

Decomposition : architectural complexity

Complexity - the number and the nature of the relationships between components in that system.

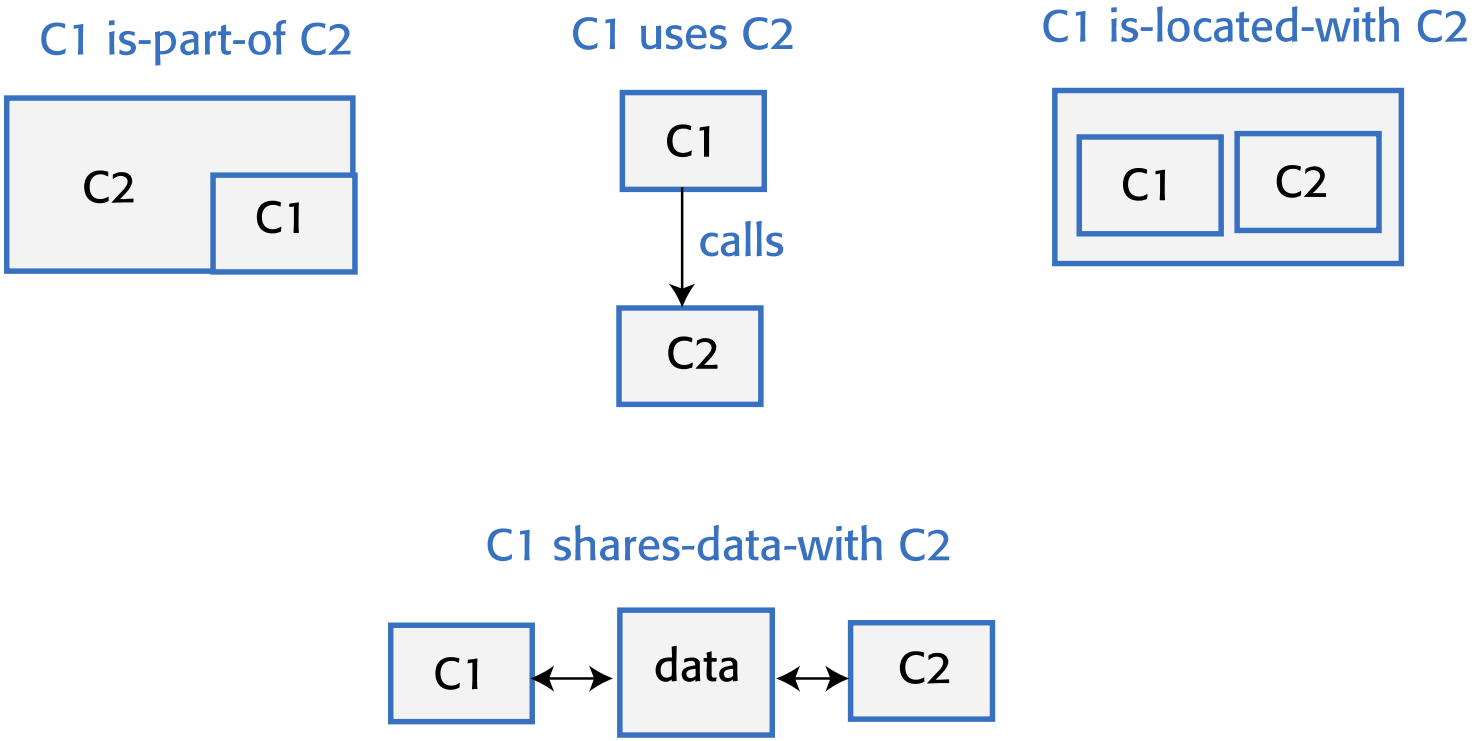
Decomposition principles

- *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.
- *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.

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

Figure 4.7 Examples of component relationships
Ian Sommerville – Engineering Software Products

Component organization



Architectural design

The architectural design is normally expressed as a block diagram presenting an overview of the *system structure*.

More specific models showing how sub-systems *share data*, are *distributed*, and *interface* with each other, are necessary.

Modules and components

A ***module*** is an *implementation unit*.

Modularization is realised based on *functionality*, generally aiming at *functional independence* (strong cohesion and low coupling).

A module ***defines*** behavior and interaction with other modules.

A ***component*** is an *execution unit.*

Modules and components

A ***module*** is an *implementation unit*.

A ***component*** is an *execution unit.*

Components are instantiated at runtime based on the definitions in modules.

Components have the *behavior defined in the modules* they are instantiated from, and *interact through connectors*.

Defining the ***dynamic structure*** of the system (components and connectors) generally aims at fulfilling the *requirements for quality attributes*.

Architectural perspectives

Static perspective **-** shows *implementation units* of the system (modules), defines their *interfaces* and shows the *relationships* among them.

Dynamic perspective (process) – shows the *runtime structure* of a system composed of *execution units* (components) and *relationships* among them (connectors).

Deployment perspective – shows how the implementation units are deployed on the infrastructure elements (hardware and software platforms).

The architectural project is ***documented*** from ***more perspectives*** using appropriate *viewtypes.*

Topics covered

Software architecture

Architectural design

Architectural styles

Architectures for software products

Architectural styles

Architectural style - *generic architectural model*, specific to an architectural perspective.

Utiliy – starting point in defining architectures for particular systems.

Obs. Most large systems are heterogenous and do not follow a single architectural style.

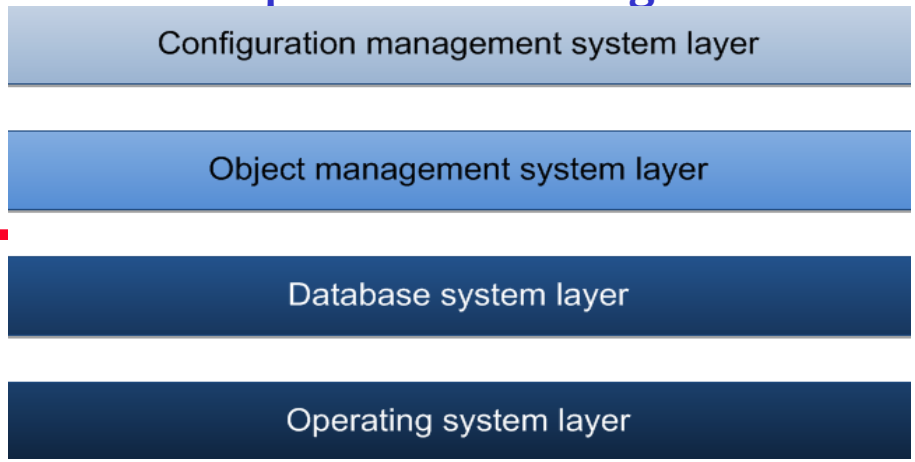
Examples:

- Static perspective : layered
- Dynamic perspective : pipe-and-filter, repository, client-server, event-based

Abstract machine style (layered)

Organizes the ***implementation units*** of the system on more ***abstraction levels***.

Example: Version management



Layers – libraries or collections of related services.

Each layer offers a set of services accesible through an ***interface*** (ex. API).

Interaction takes place only between ***adjacent*** layers, a high-level layer accessing the services offered by the low-level layer.

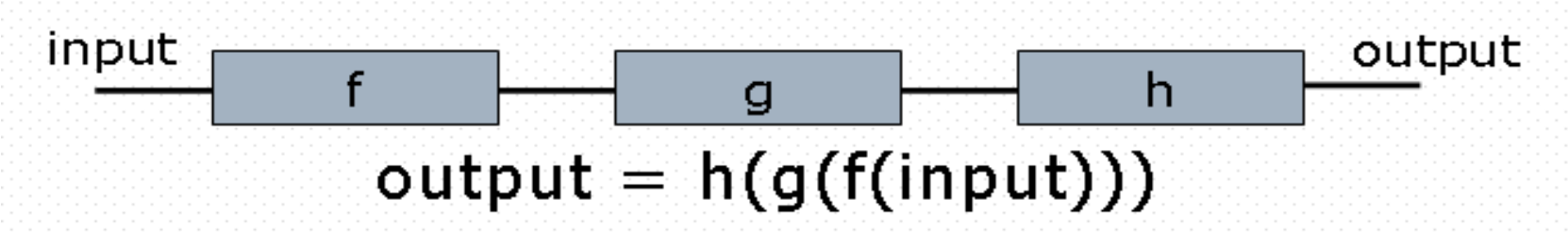
Supports ***incremental development*** of the sub-systems on different layers. Changes behind interfaces are invisible to other layers and when the interface of a layer is changed, only the adjacent layer is affected.

Pipe-and-filter style

Organizes the system as a series of components (filters) connected by pipes.

Components are processes that realize *functional transformations* on *data in the input flow* and send the result in the *output flow*.

A component *does not need to wait* for all the data from the predecesor process; it starts as soon a data are available in its input flow.



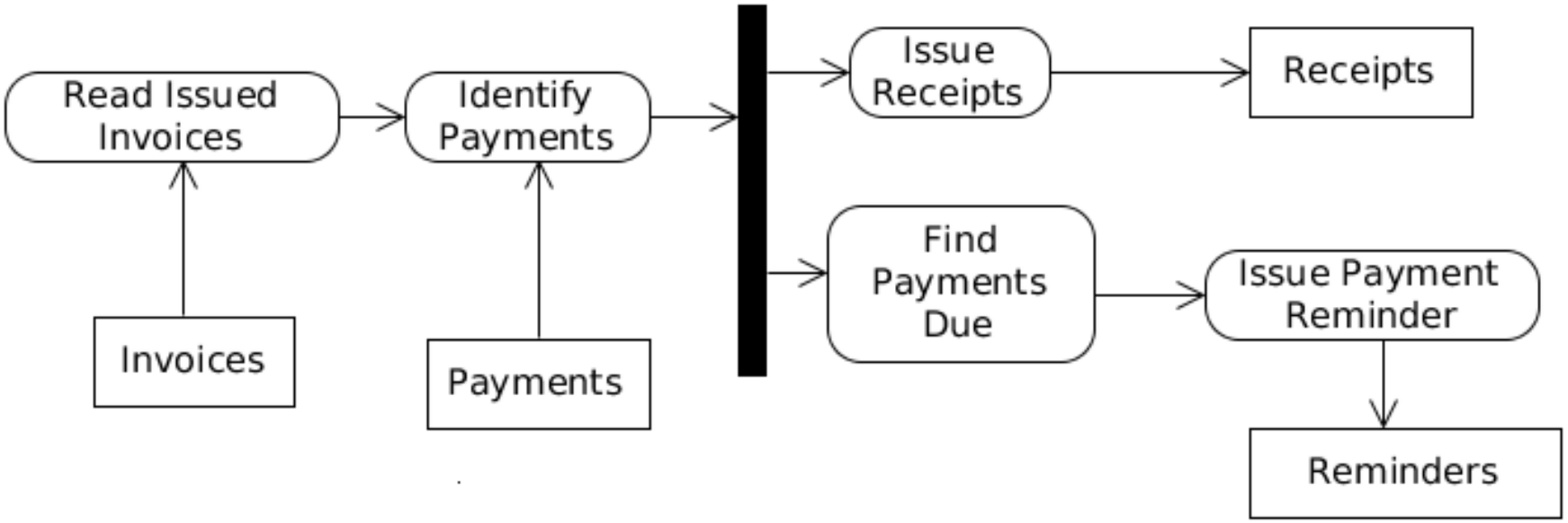
Examples :

Combining Unix commands

```
cat/etc/passwd | grep "joe" | sort > junk
```

Applications for processing audio and video streams

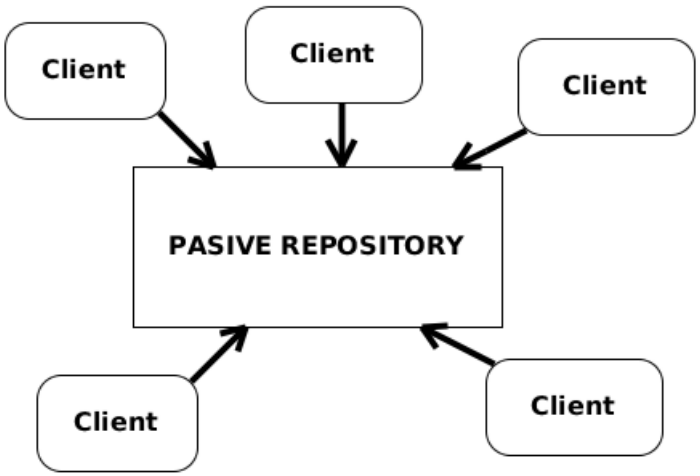
Example : Processing invoices payment

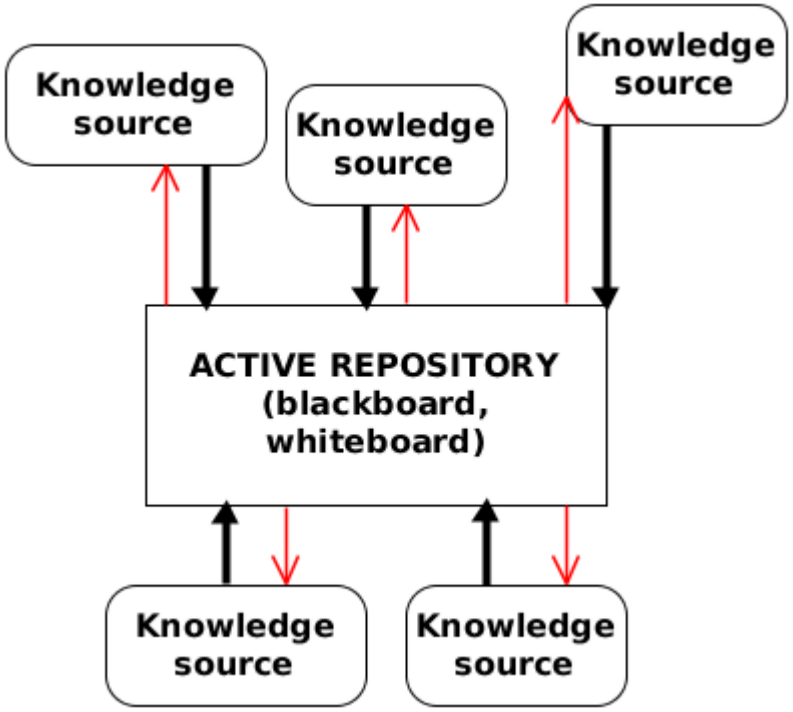


Repository style

Structures the system in *independent components* that communicate only through data stored in a *central repository*;

Repozitory style is used in most of the cases when large data amounts must be shared.

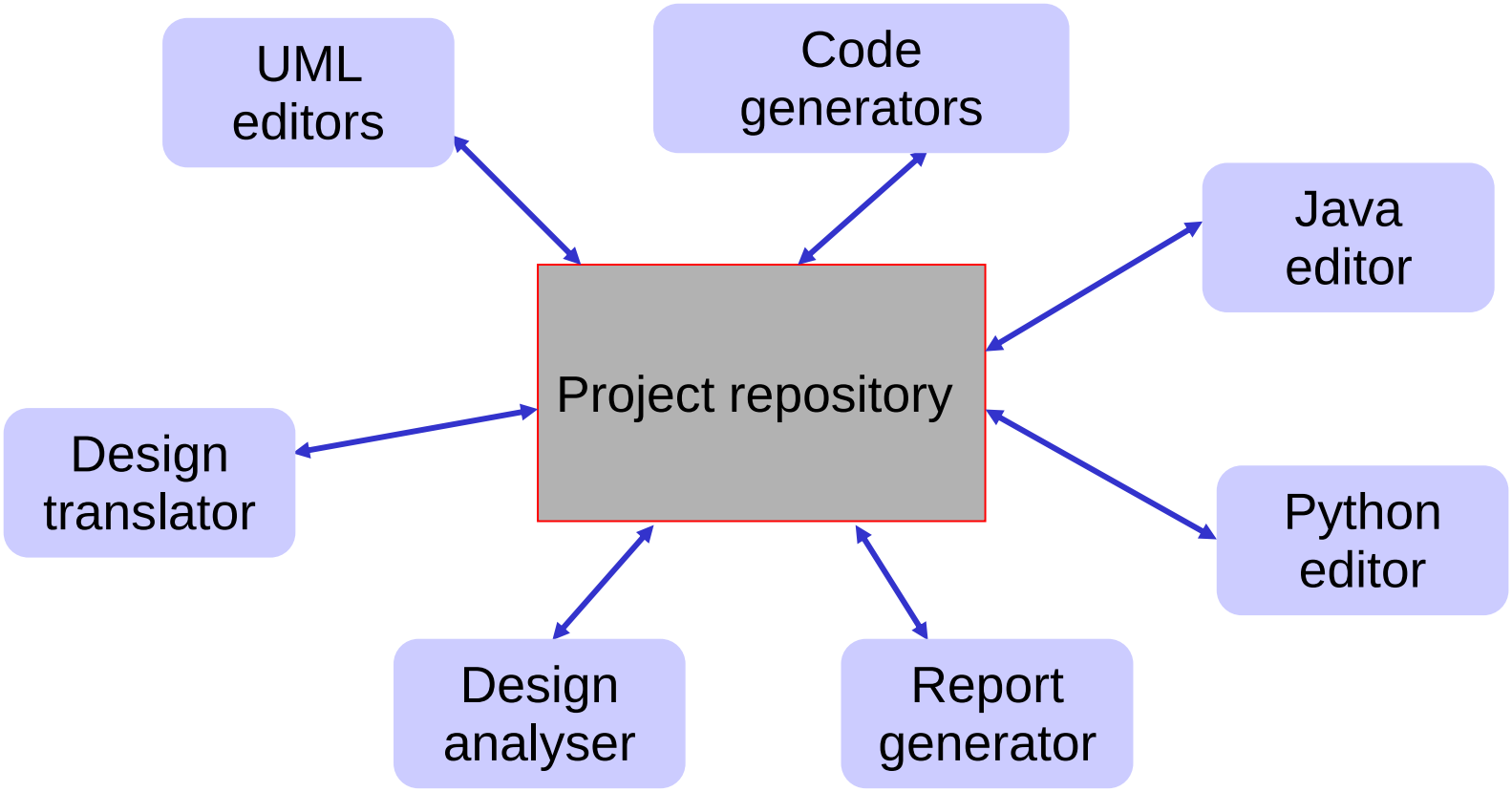




Examples:

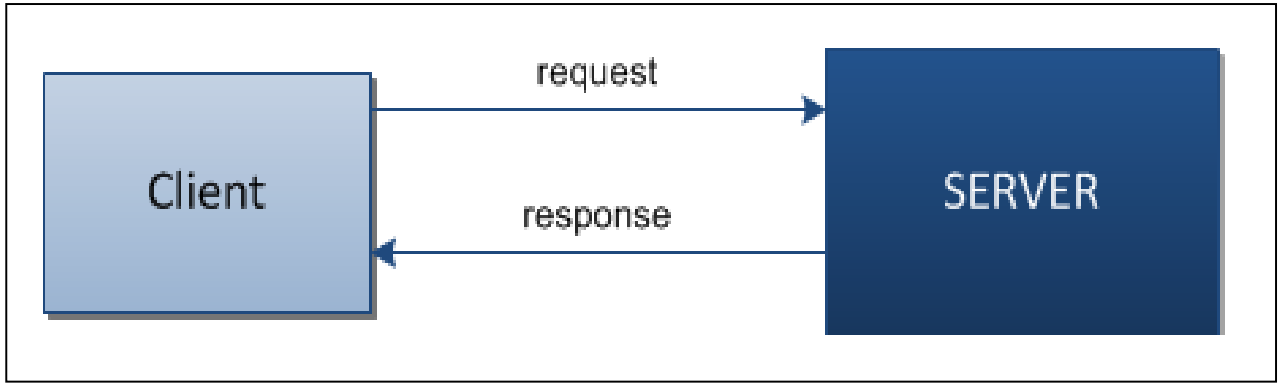
Databases – passive repository Artificial intelligence – active repository

A repository architecture for an IDE



Client – server style

Organizes the system in *client* components that *request services* and *server* components that *provide services*.



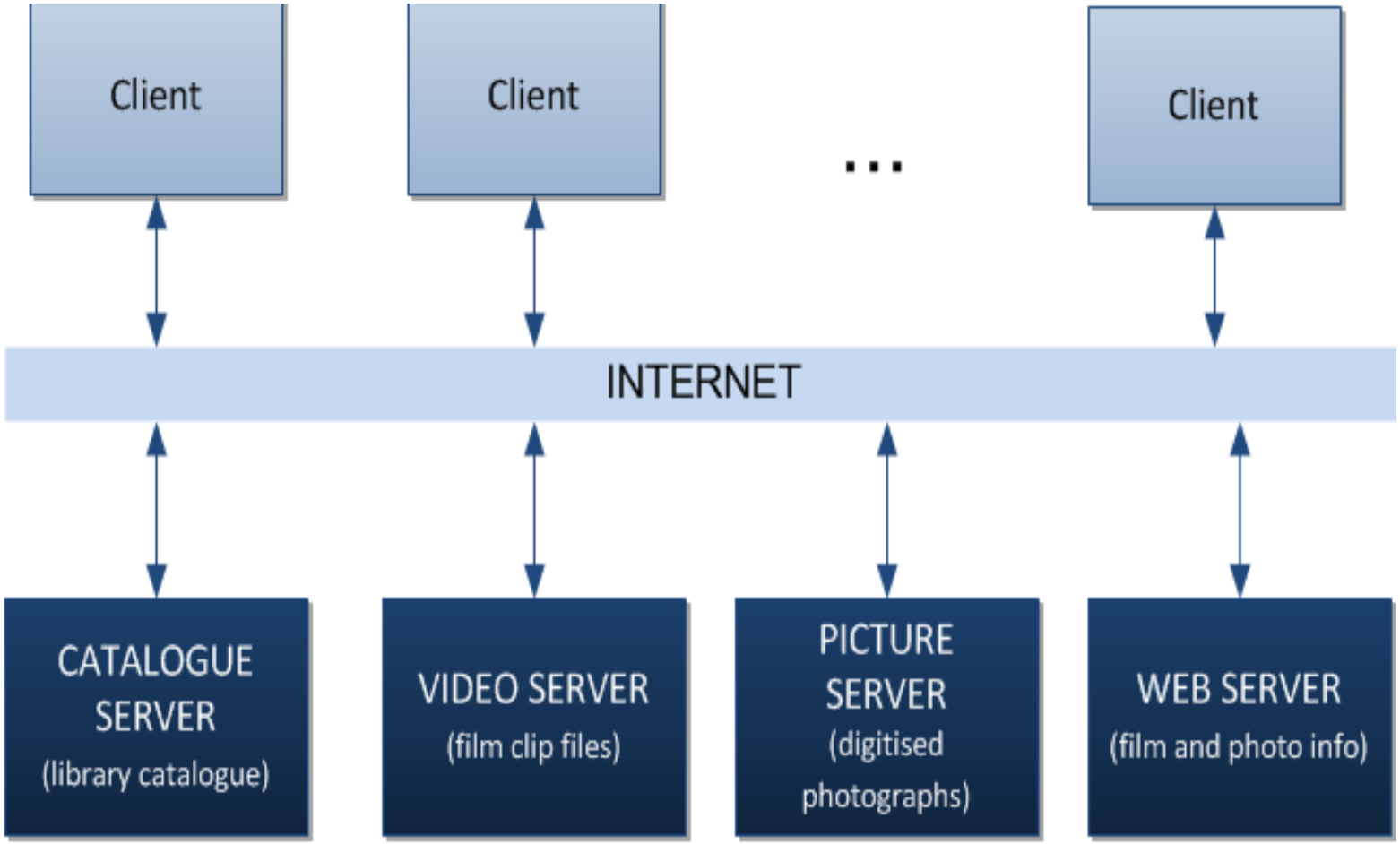
Distributed system model (shows how data and processing is distributed across a range of components):

Set of stand-alone *servers* which provide specific services such as printing, data management, etc.

Set of *clients* which call on these services.

Network which allows clients to access services.

Example: Film and picture library

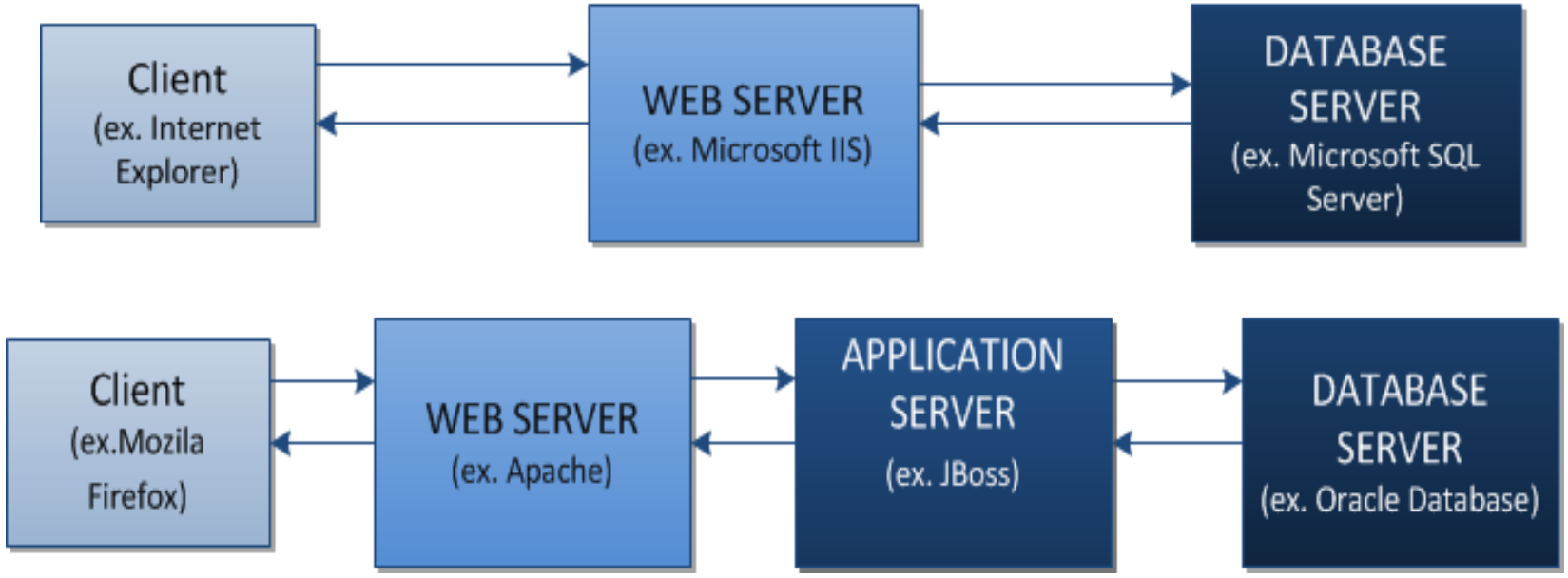


Extended client-server style

Classic client –server : 2 tiers

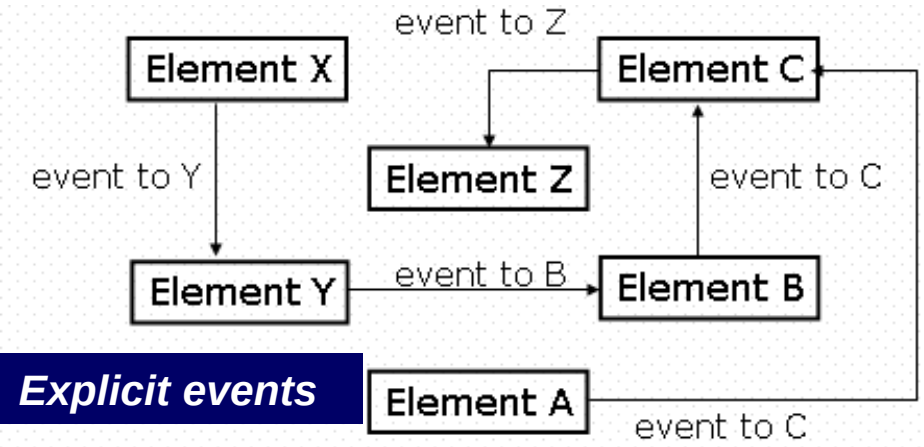
Extended client – server : 3, 4, ..., n tiers

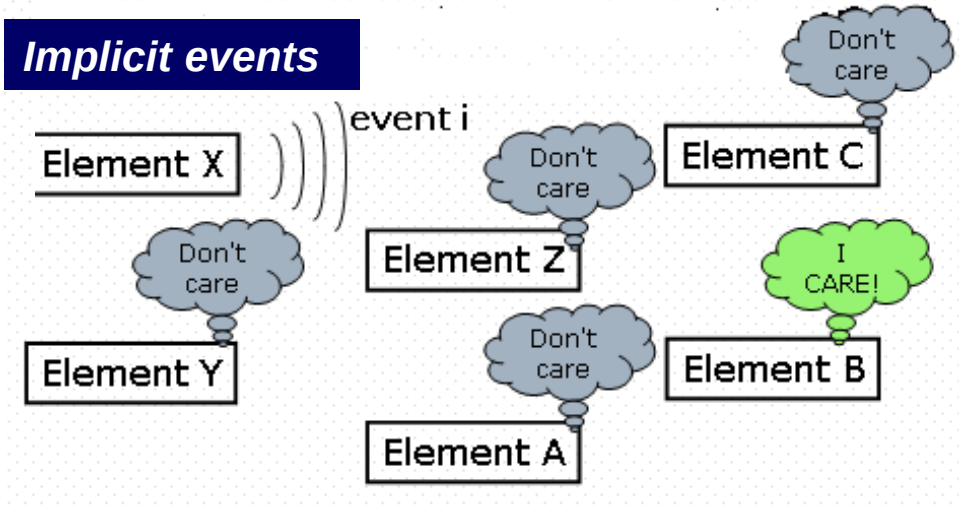
Structures the system in client and server components on *more tiers*.



Event based styles

Organize the application in components that *generate events* and/or *react to events* generated by other components.





Examples:

GUI libraries – interactive systems

Distributed systems – av. decoupling and reorganising

Formative evaluation

1. Which are the main architectural perspectives and what shows each of it ?

2. Realize the mapping between the architectural style and how is organized the system that conforms to the style.

https://forms.gle/evD5Qboz9M4tXJE29

Topics covered

Software architecture

Architectural design

Architectural styles

Architectures for software products

Generic application architectures

Application systems are designed to meet an organizational need.

As businesses have much in common, their application systems also tend to have a *common architecture* that reflects the application requirements.

A *generic architecture* is *configured* and *adapted* to create a system that meets specific requirements.

Use of application architectures

As a starting point for architectural design.

As a design checklist.

As a way of organizing the work of the development team.

As a means of assessing components for reuse.

As a vocabulary for talking about application types.

Design guidelines and software architectures

Figure 4.8 Architectural design guidelines
Ian Sommerville – Engineering Software Products

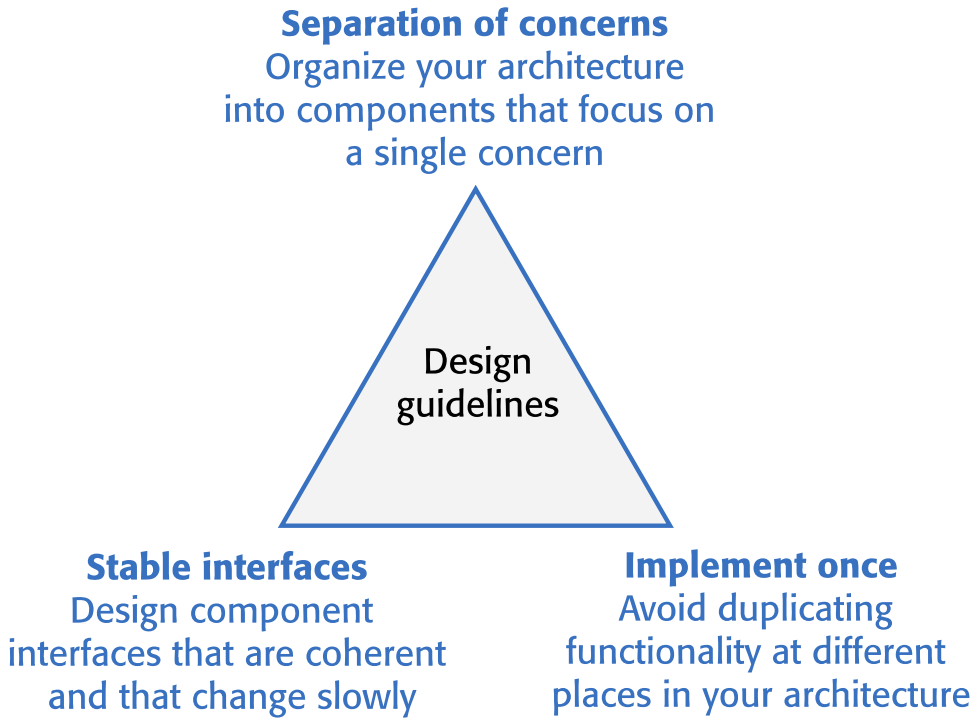


Figure 4.6 An architectural model of a document retrieval system
Ian Sommerville – Engineering Software Products

Layered architecture : example

Separation of concerns on layers :

- user interaction
- user interface management
- information retrieval, etc.

Independent units in layer, not overlapping in functionality.

- lower layers - provide general functionality used on higher levels.

Ideally, units at level X should only interact with the APIs of the units in level X-1.

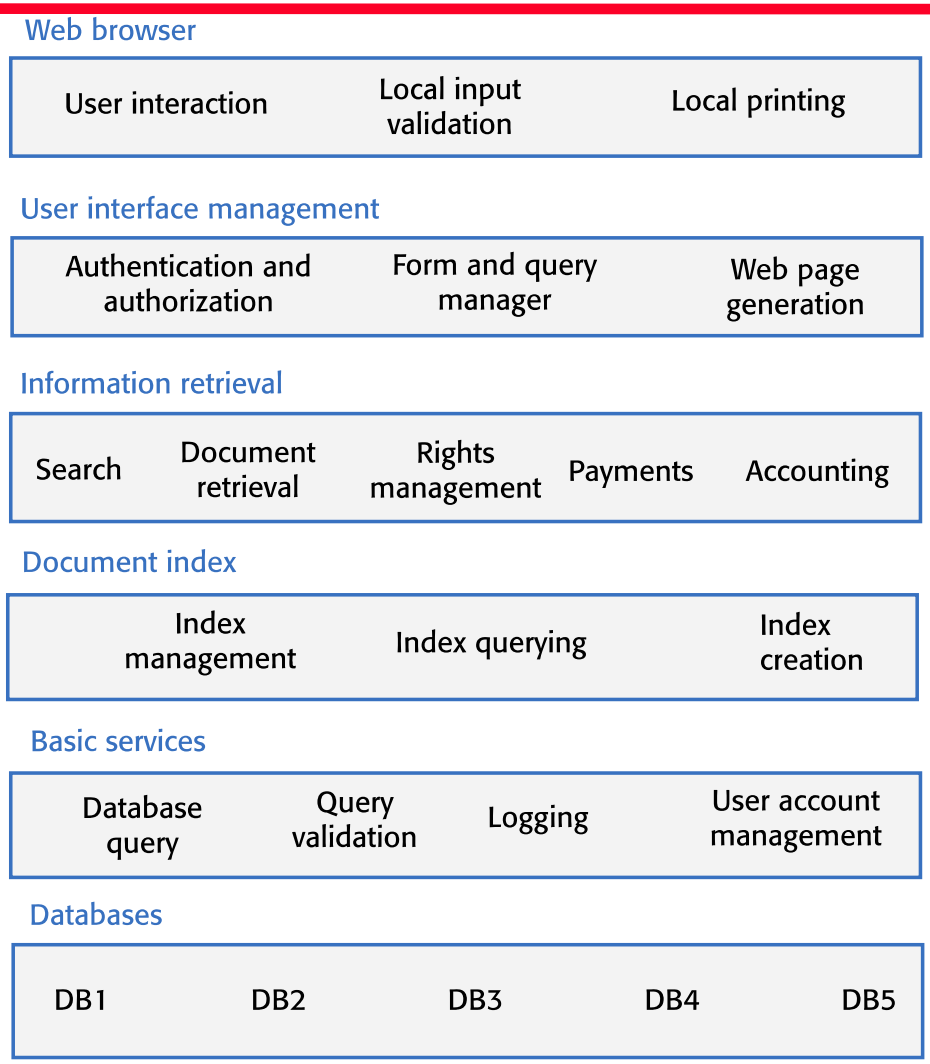
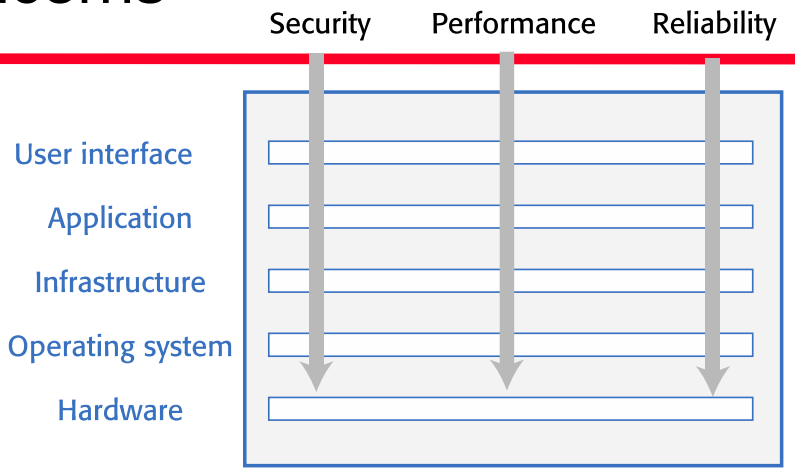


Figure 4.9 Cross-cutting concerns
Ian Sommerville – Engineering Software Products

Cross-cutting concerns



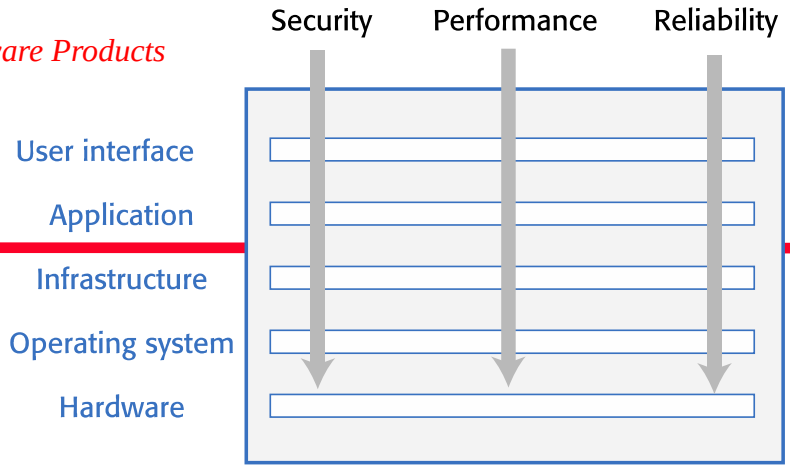
Cross-cutting concerns = systemic concerns

affect the whole system.

- Affect :
  - all layers in the system
  - the way in which people use the system.
- Are completely different from the functional concerns represented by layers.
- Generate inevitably interactions between the layers that must take them into account.
- Difficult to modify the system, after it has been designed, to improve a cross-cutting concern (ex. security).
-

Figure 4.9 Cross-cutting concerns
Ian Sommerville – Engineering Software Products

Cross-cutting concerns: security



Attackers can try to use of vulnerabilities in different technologies used in different layers (ex. SQL database, Firefox browser).

⇓

Protection from attacks at each layer : at lower layers in the system, from successful attacks that have occurred at higher-level layers.

Single security level = critical system vulnerability if it stops working properly or is compromised in an attack.

Distributed security across the layers ⇒ more resilient system to attacks and software failure.

Layered functionality in a web-based application

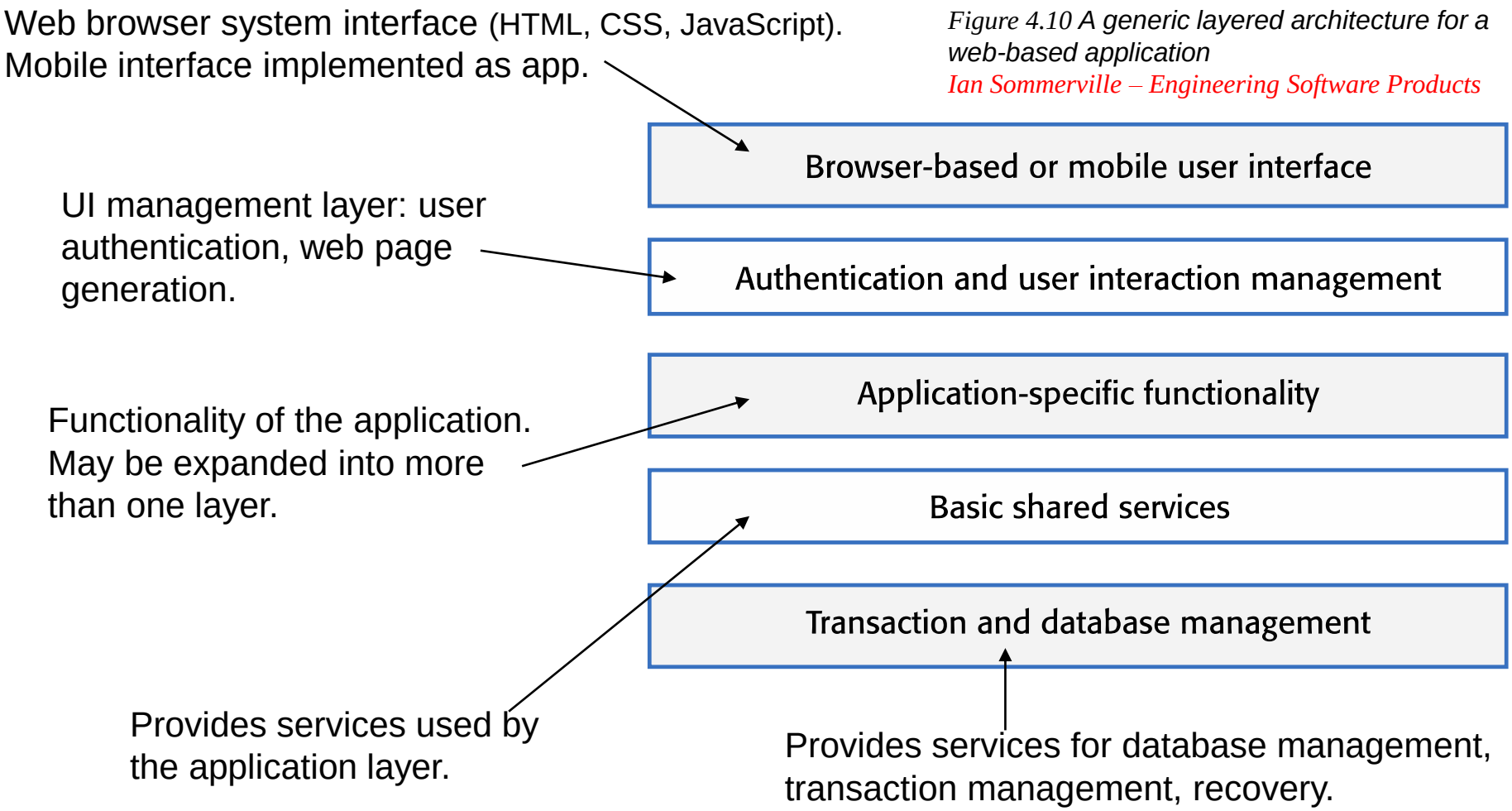


Figure 4.10 A generic layered architecture for a web-based application
Ian Sommerville – Engineering Software Products

iLearn architectural design principles

iLearn system goal *: adaptable, universal* system that could be easily updated as new learning tools became available.

Design principles

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 hard-wired 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 age-appropriate 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.

Figure 4.11 A layered architectural model of the iLearn system
Ian Sommerville – Engineering Software Products

Designing iLearn as a service-oriented system

- Principles led 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.
    -

Designing iLearn as a service-oriented system

From static perspective, the units of implementation are organized in a layered architecture.

Implementation units in layers are instantiated at runtime as components providing services.

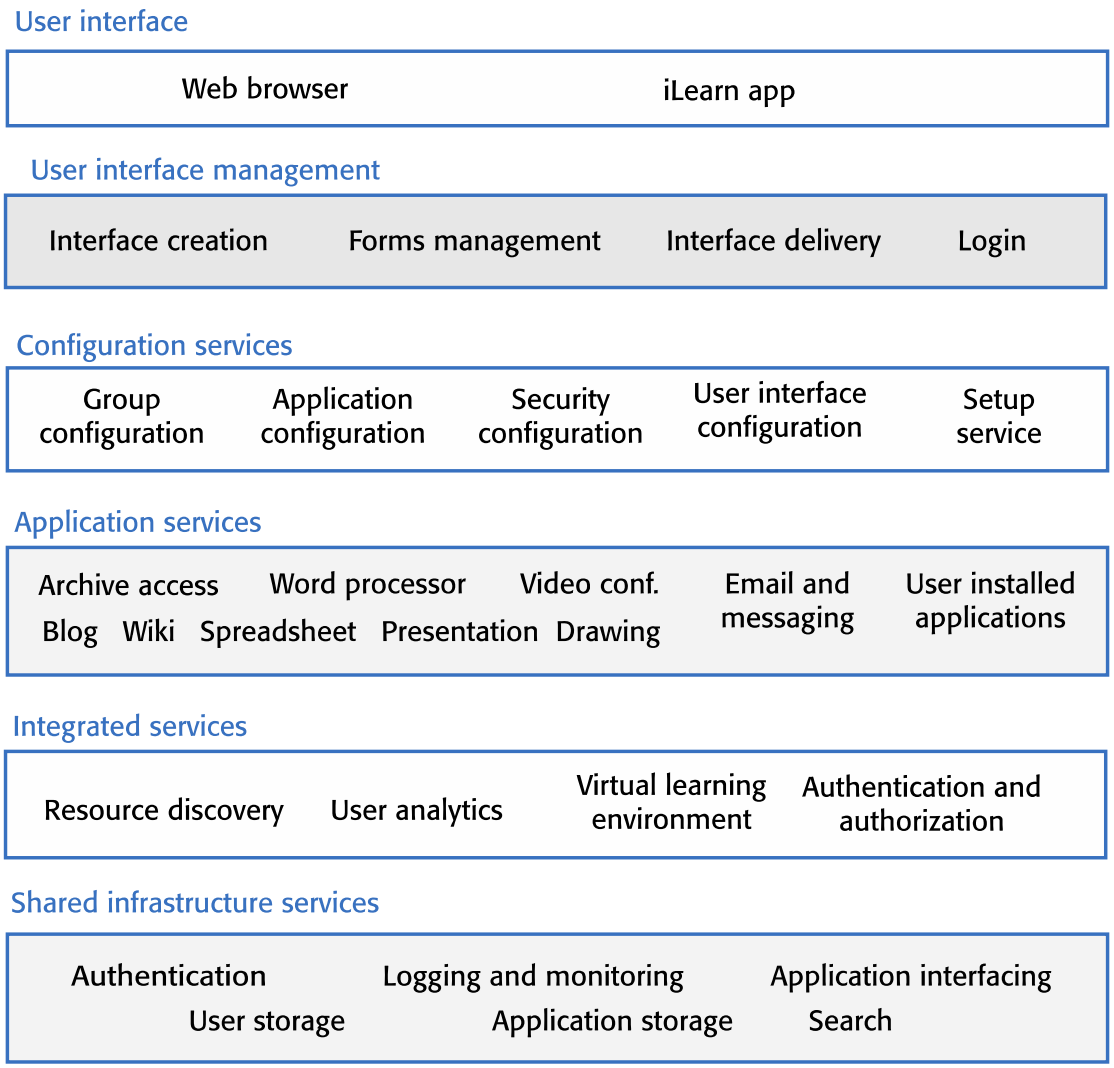


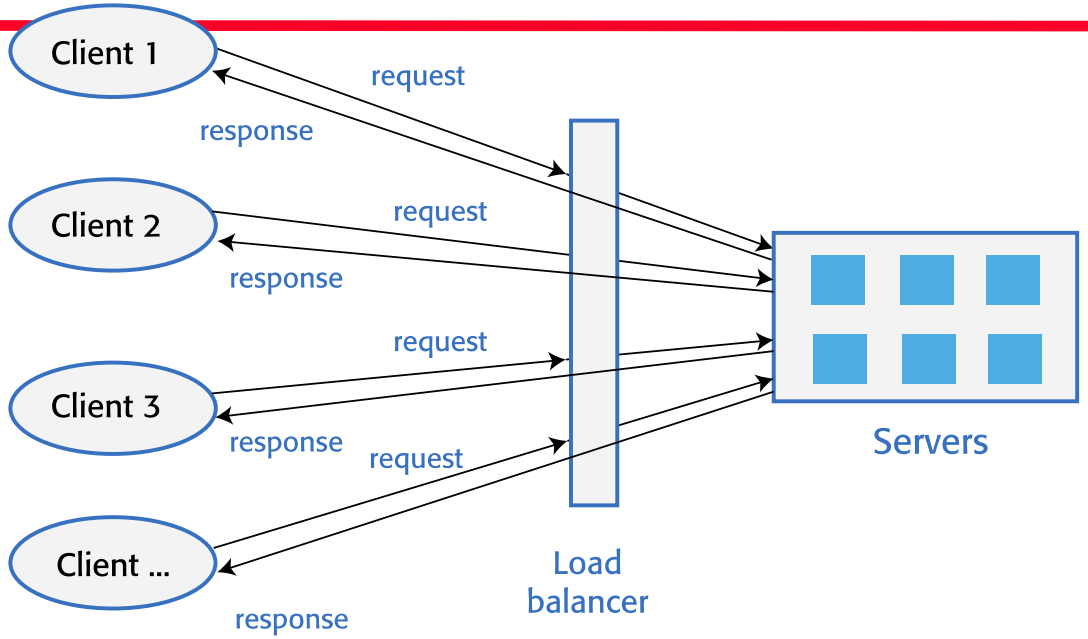
Figure 4.11 A layered architectural model of the iLearn system
Ian Sommerville – Engineering Software Products

Figure 4.12 Client-server architecture
ral model Ian Sommerville – Engineering Software Products

Distribution architecture

The distribution architecture of a software system defines:

- servers in the system
- allocation of components to servers.



Client-server architectures are suited to applications where clients access a shared database and business logic operations on that data.

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 communication

Client-server web 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.

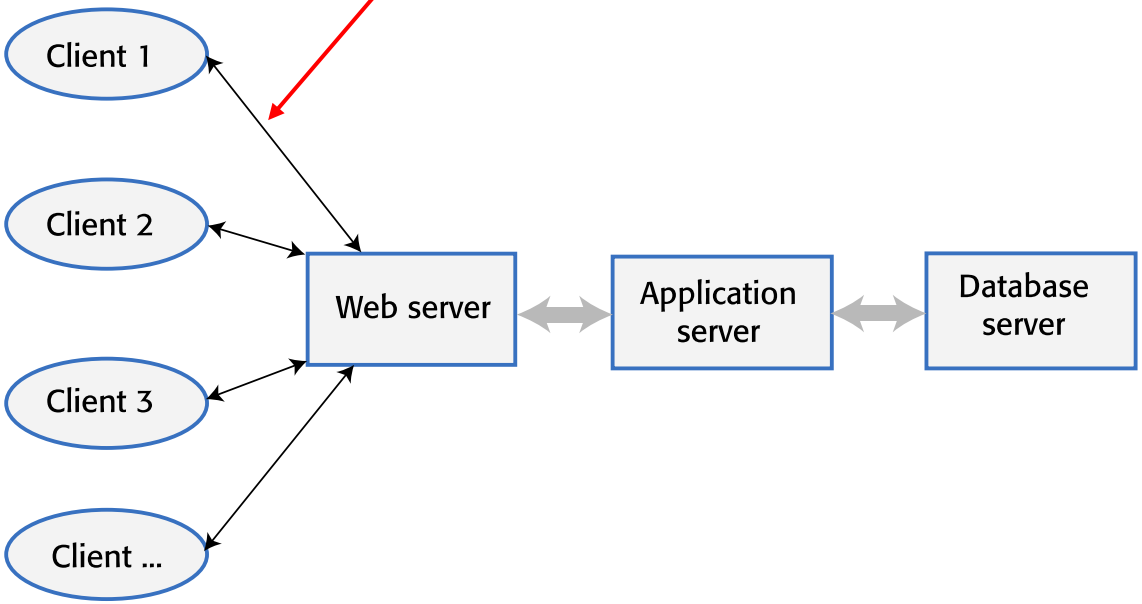


Figure 4.14 Multi-tier client-server architecturral model
Ian Sommerville – Engineering Software Products

Ways of representing structured text data transferred through HTTP protocol

- 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.

Service-oriented architecture

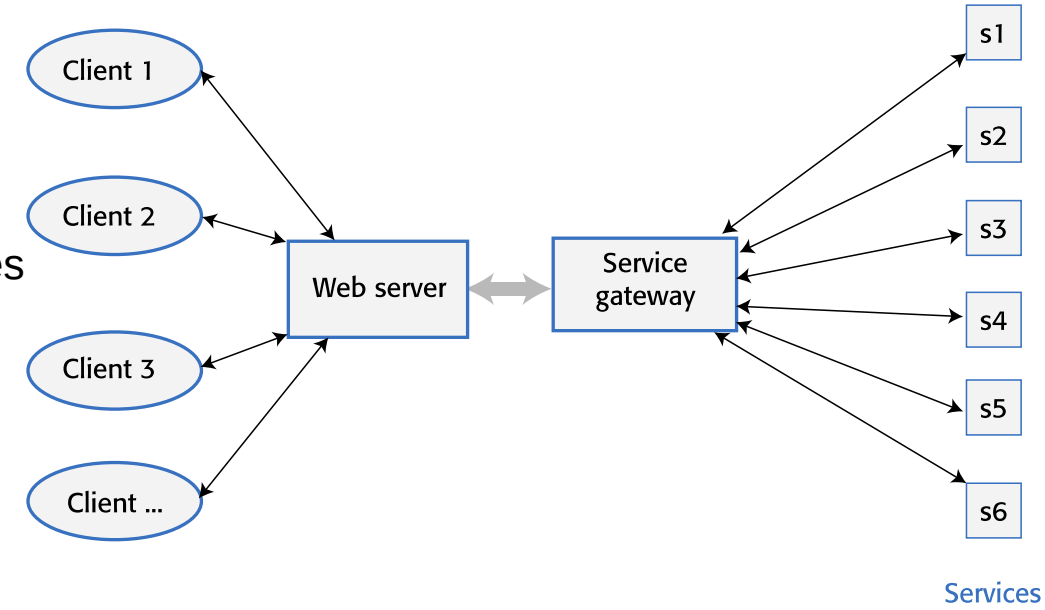
Service - oriented architecture

Services - stateless components ⇒ can be replicated and can migrate from one computer to another.

Many servers may be involved in providing services.

- Easy to scale as demand increases
- Resilient to failure.

Figure 4.15 service-oriented architecturral model
Ian Sommerville – Engineering Software Products



Issues in architectural choice

- Data type and data updates

  Structured data *shared* by different system features ⇒ a single shared database that provides locking and transaction management.

  Data *distributed* across services ⇒ necessity to keep data consistency adds overhead to the system.

- Change frequency

  Anticipated regularly changes or replaces ⇒ isolate components as separate services.

- The system execution platform

Execution platform	Application type	Software architecture
Cloud	accessed over the Internet	service-oriented (scalable)
Local servers	business system	multi-tier

Technology choices

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

Kinds of database commonly used:

- Relational databases - data organized into structured tables
- NoSQL databases - data has a flexible, user-defined organization.

Relational databases (ex. MySQL) - suitable for situations where you need transaction management and the data structures are predictable and fairly simple.

NoSQL databases (ex. MongoDB) - more flexible and potentially more efficient than relational databases for data analysis.

- NoSQL databases - data organized hierarchically rather than as flat tables $\Rightarrow$ more efficient concurrent processing of ‘big data’.
-

Delivery platform

Delivery platform options:

- web-based,
- mobile product,
- both

Mobile issues:

Intermittent connectivity $\Rightarrow$ provide a limited service without network connectivity.

Processor power $\Rightarrow$ minimize computationally-intensive operations.

Power management $\Rightarrow$ minimize the power used by your application.

On-screen keyboard $\Rightarrow$ minimize input using the screen keyboard

Good practice - separate browser-based and mobile versions of the product front-end.

Obs. It is possible to need a completely different decomposition architecture in these different versions to ensure that performance and other characteristics are maintained.

Server

Key decision : to run on *customer servers* or on *the cloud.*

Recommendation :
Consumer products that are not simply mobile apps – developed for the cloud.

Business products: difficult decision.

- cloud security concerns $\Rightarrow$ run on in-house servers.
- not predictable pattern of system usage $\Rightarrow$ run on cloud OR need to design the system to cope with large changes in demand

Important choice - which cloud provider to use.

Open source

Open source software - software that is freely available and modifiable.

- Advantage : Allow to reuse $\Rightarrow$ reduced development costs and time to market.
- Disadvantages : added constraints and no control over its evolution.

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

Categories of development technologies:

- mobile development toolkit
- web application framework

Developer needs to conform to built-in assumptions about system architectures.

Development technology may have an indirect influence on the system architecture.

Obs. 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.

Formative evaluation

1. What is the meaning of cross-cutting concerns in layered architectures?

2. Suppose a mobile platform is selected for delivering a software application. Specify the main problems specific to this type of platform and their solutions.

https://forms.gle/pLjfwkQmu7mv6j5Q8

Key points

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.

Architecture design is driven mainly by the requirements for quality attributes. Architectural design implies a series of decisions in order to realize the best trade-off in fulfilling quality requirements which may be in conflict.

Architectural design involves defining system interfaces with its context, decomposing the system in elements that interact in order to fulfill functional and quality requirements, and establishing relationships (interfaces) among these elements.

Ways to minimize complexity are separation of concerns, avoiding functional duplication and focusing on component interfaces.

Key points

Software architecture is represented from multiple perspectives: static, dynamic and deployment perspective.

Examples of architectural styles are, from dynamic perspective, repository style and client-server style, and abstract machine style from static perspective.

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

The distribution architecture defines the organization of the servers and the allocation of components to these servers.

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.