Software architecture

Antonio Brogi

Department of Computer Science
University of Pisa

GLOBAL EDITION
UNPREPARED
TO DO
ANALYSIS
DEV
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville
P

Software architecture

To create good products we need to pay attention to its overall organization

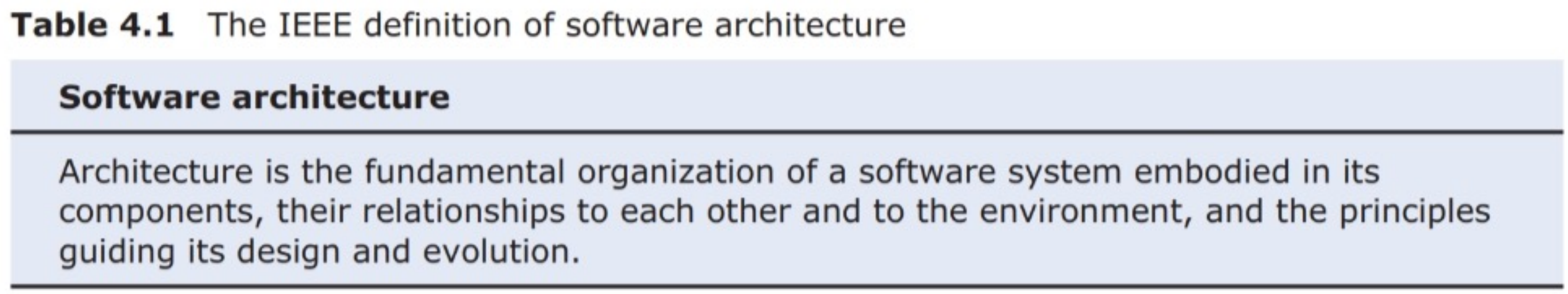


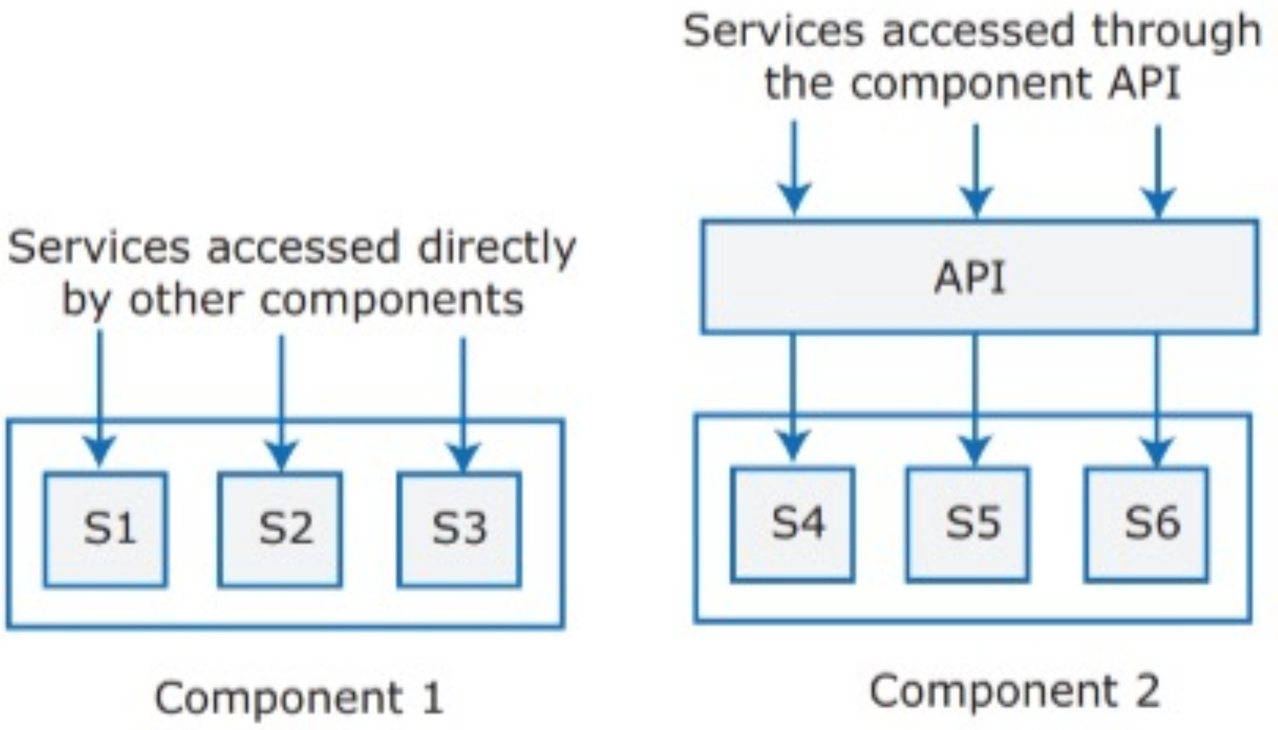
Table 4.1 The IEEE definition of software architecture

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.

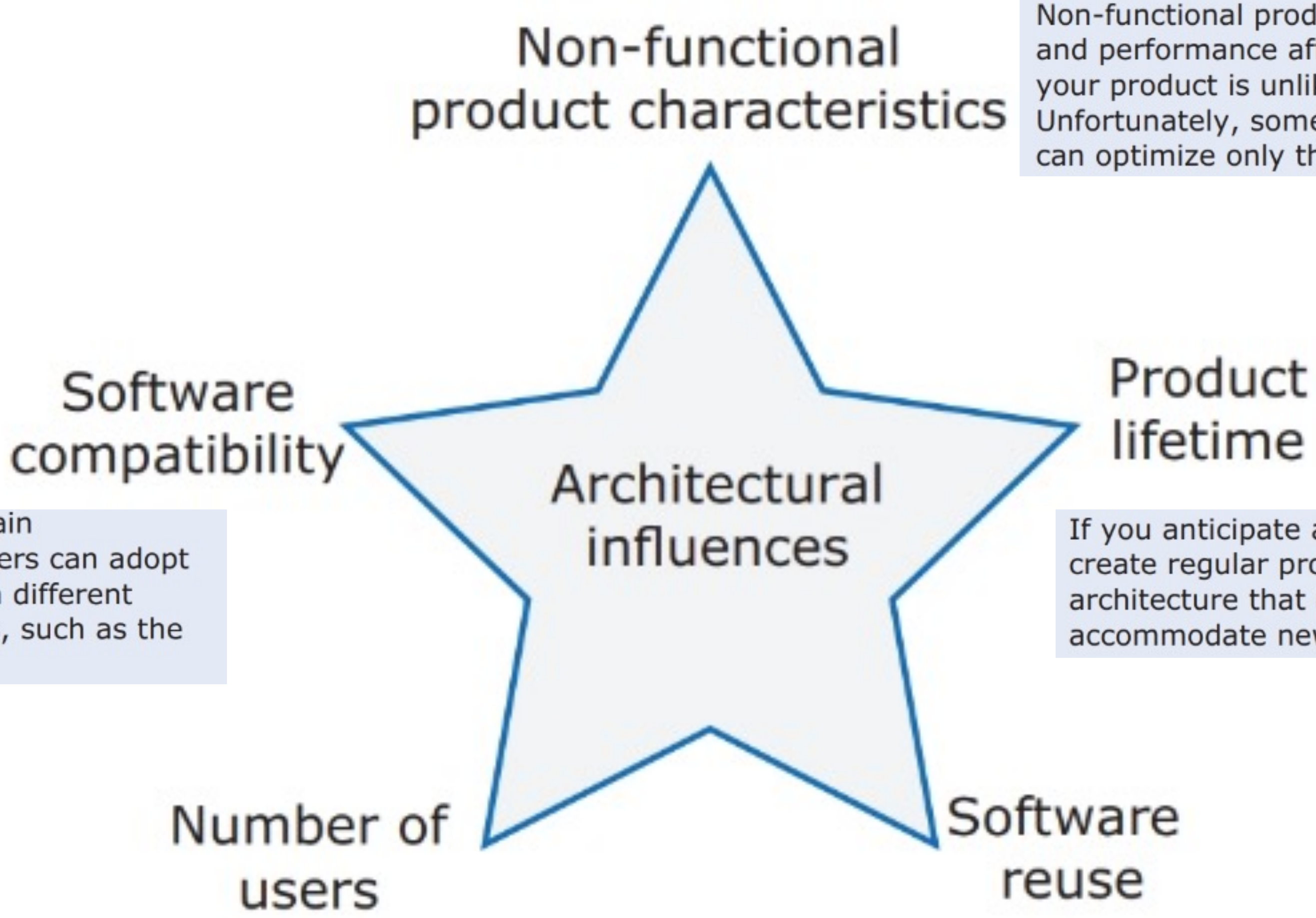
Software architecture affects performance, usability, security, reliability, maintainability, …

Component

- Element implementing a coherent set of features
- Collection of services that may be used by other components


Services accessed directly by other components
S1
S2
S3
Component 1
Services accessed through the component API
API
S4
S5
S6
Component 2

Architectural design issues



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.

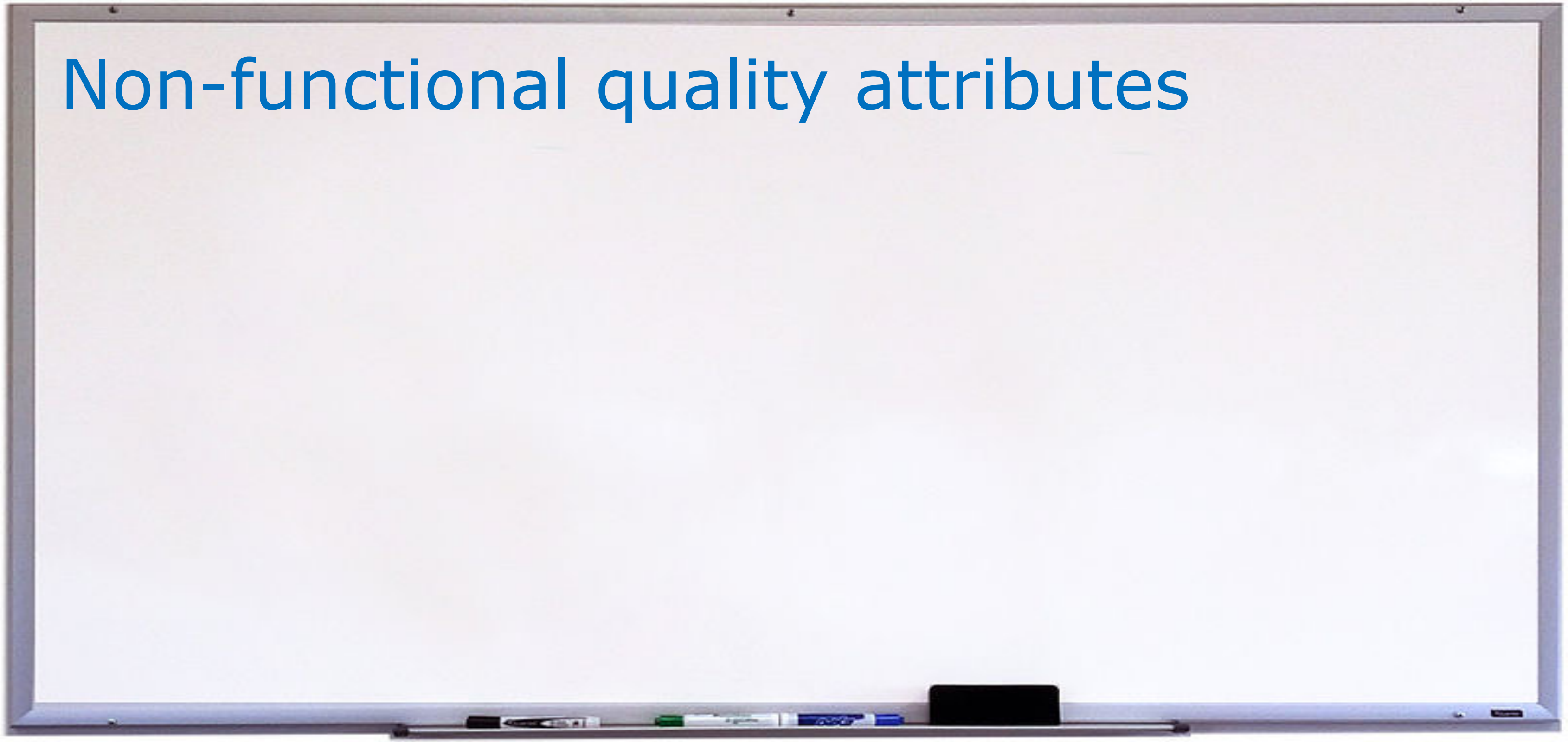
If you anticipate a long product lifetime, you need to create regular product revisions. You therefore need an architecture that can evolve, so that it can be adapted to accommodate new features and technology.

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.

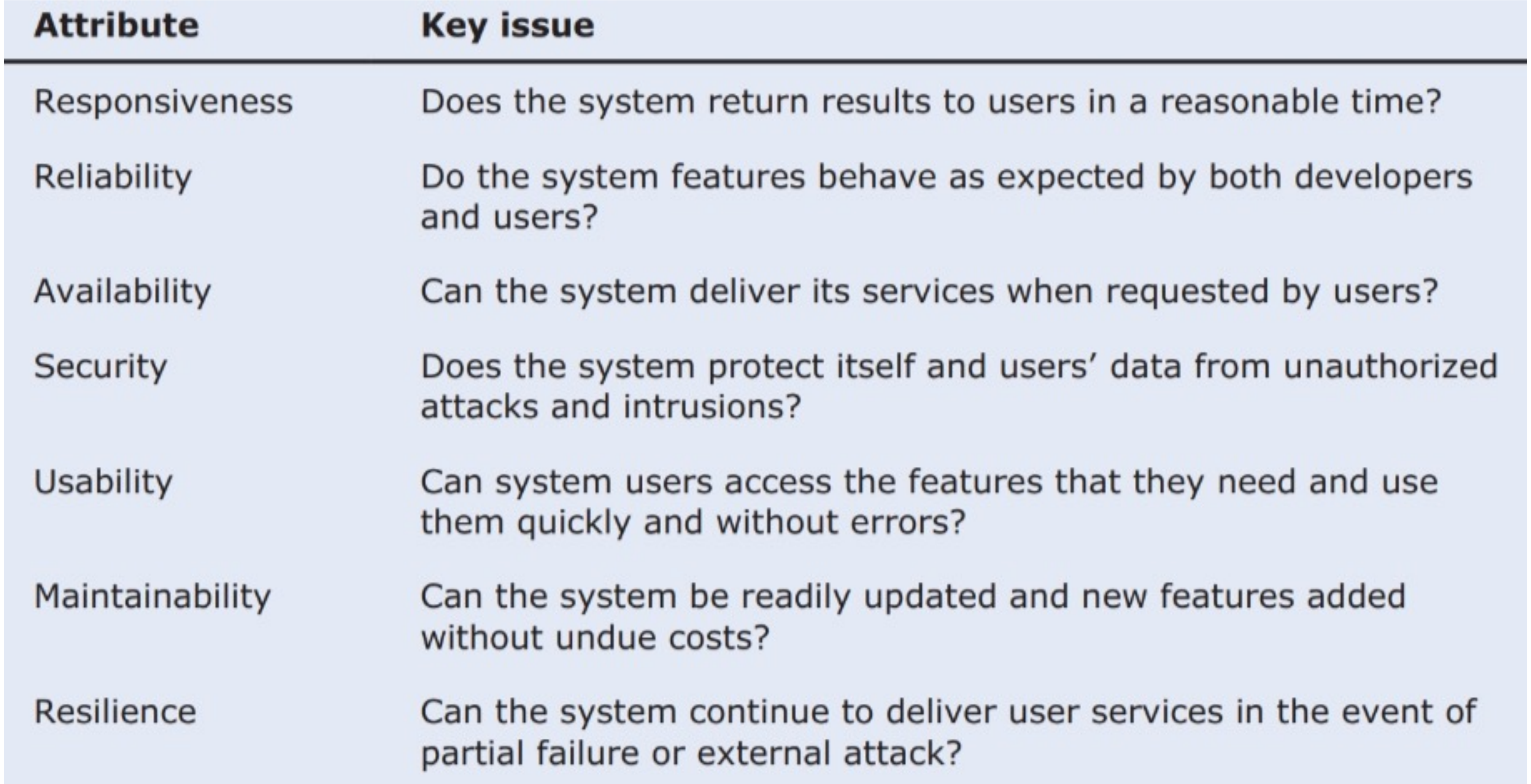
If you are developing consumer software delivered over the Internet, the number of users can change very quickly. This can lead to serious performance degradation unless you design your architecture so that your system can be quickly scaled up and down.

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.

Non-functional quality attributes



Non-functional quality attributes

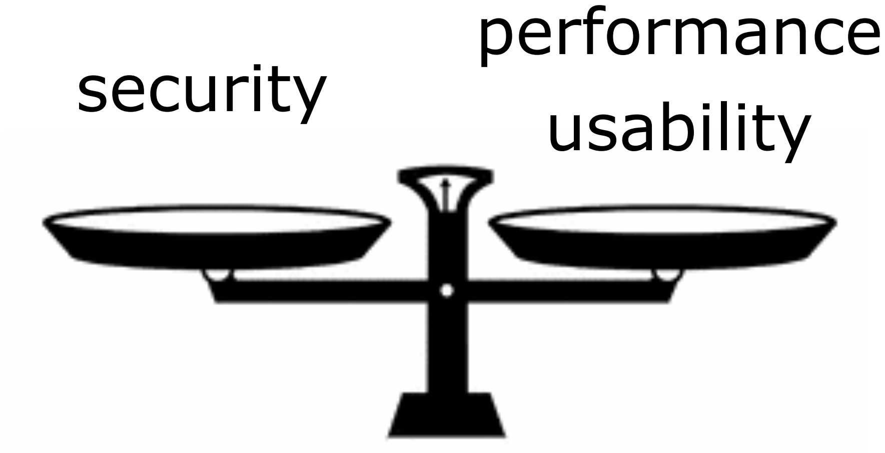


Attribute	Key issue
Responsiveness	Does the system return results to users in a reasonable time?
Reliability	Do the system features behave as expected by both developers and users?
Availability	Can the system deliver its services when requested by users?
Security	Does the system protect itself and users' 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?

(Important for final product - not for prototype)

Warning

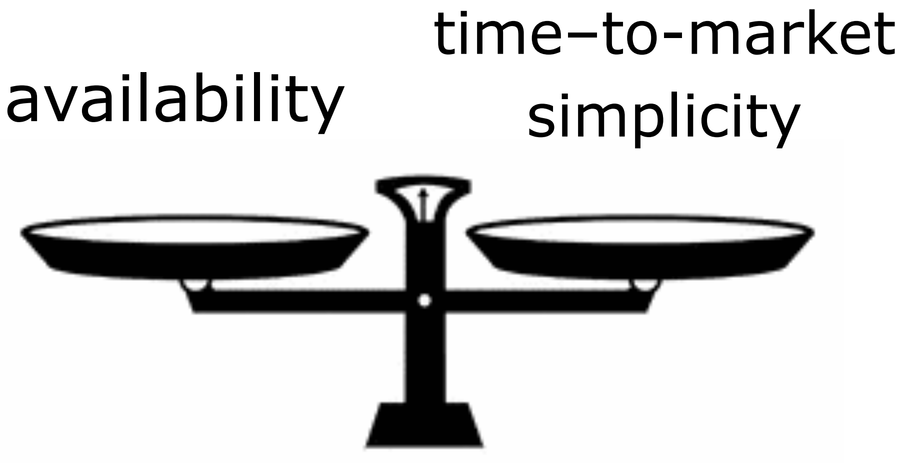
Optimizing one non-functional attribute affects others



security

performance

usability



availability

time–to-market

simplicity

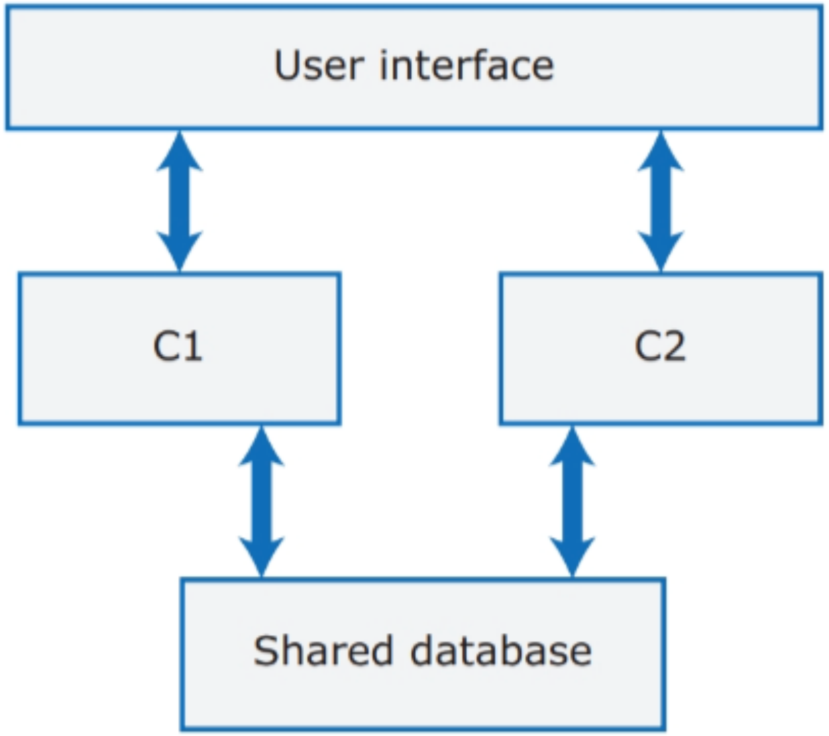
Example: Maintainability

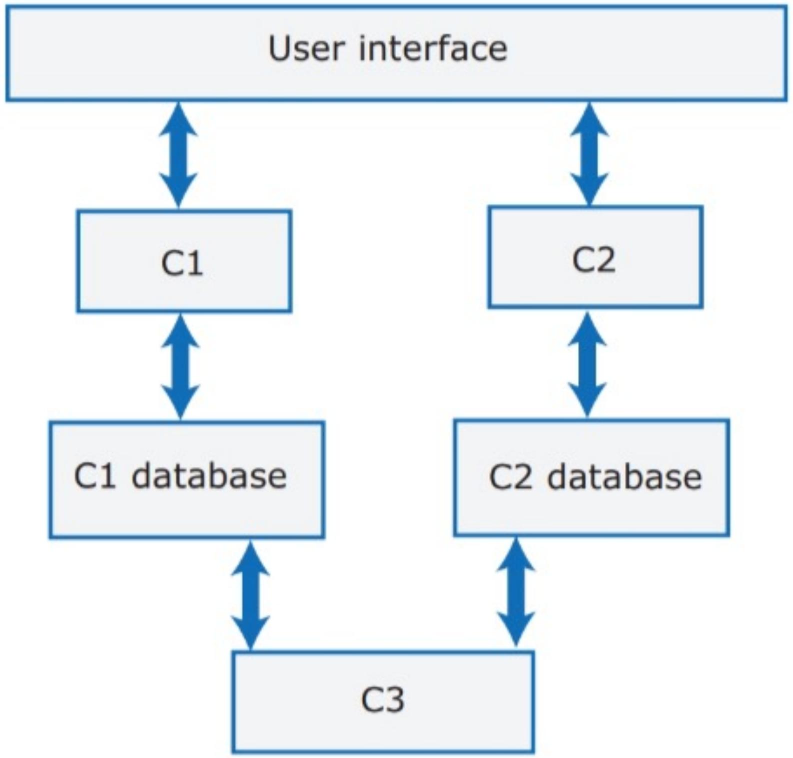
Maintainability = how difficult/expensive to make changes after release

Good practices

- Decompose system into small self-contained parts
- Avoid shared data structures

Example



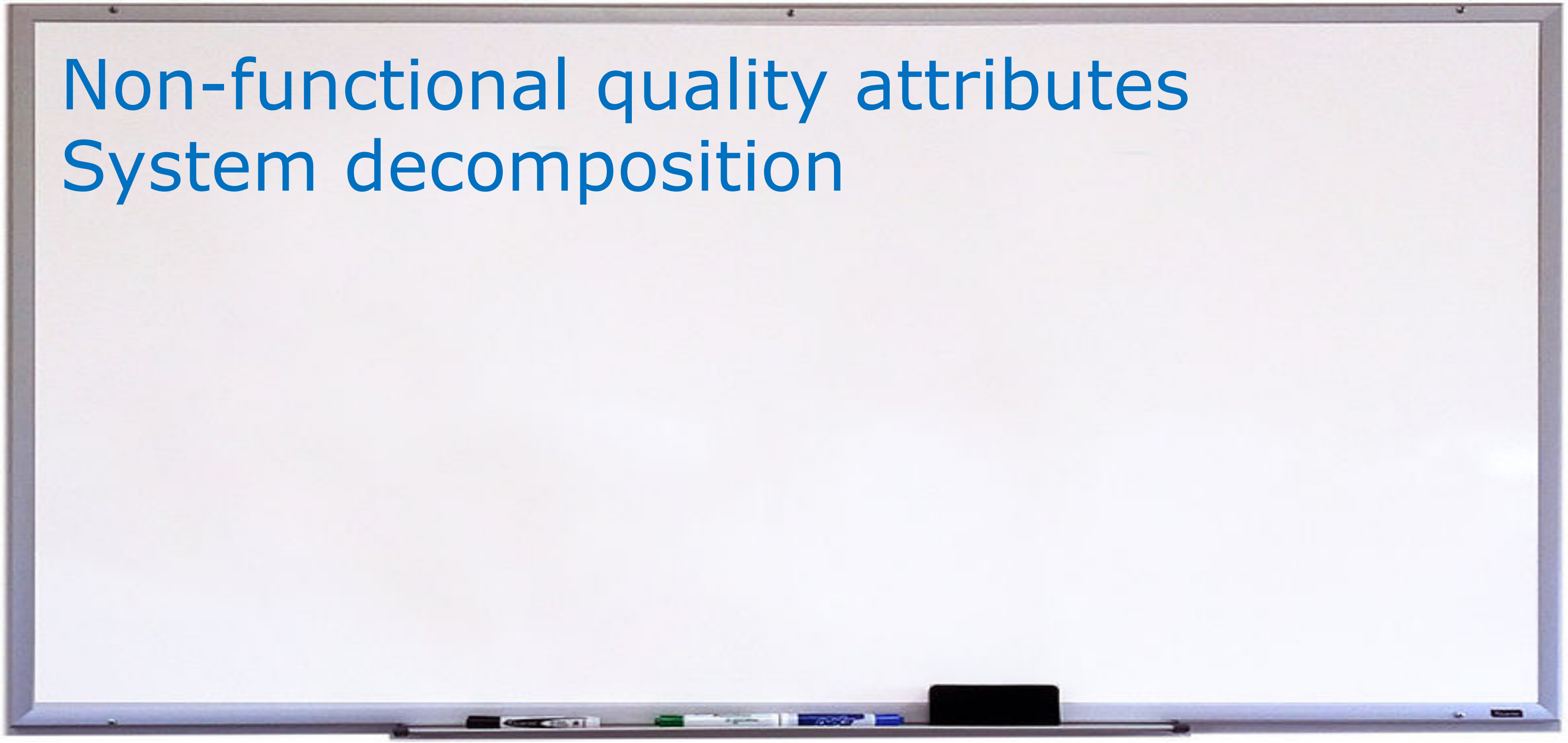


Database reconciliation

What if C1 runs slowly and needs to reorganize DB?

+ If one component needs to change the database organization, this does not affect the other component
+ System can continue to provide partial service in the event of a database failure

Cost: need of mechanisms for (eventual) data consistency

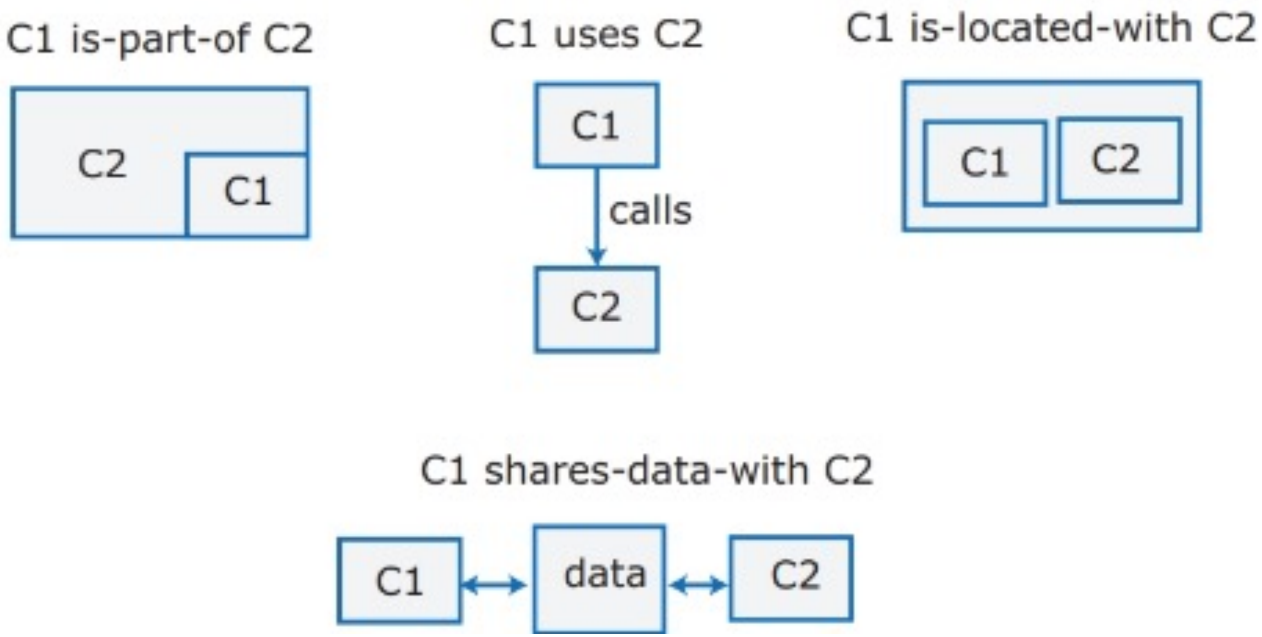


Non-functional quality attributes
System decomposition

Services, components, modules

- Service = coherent unit of functionality
- Component = software unit offering one or more services
- Module = set of components

Examples of component relationships



C1 is-part-of C2

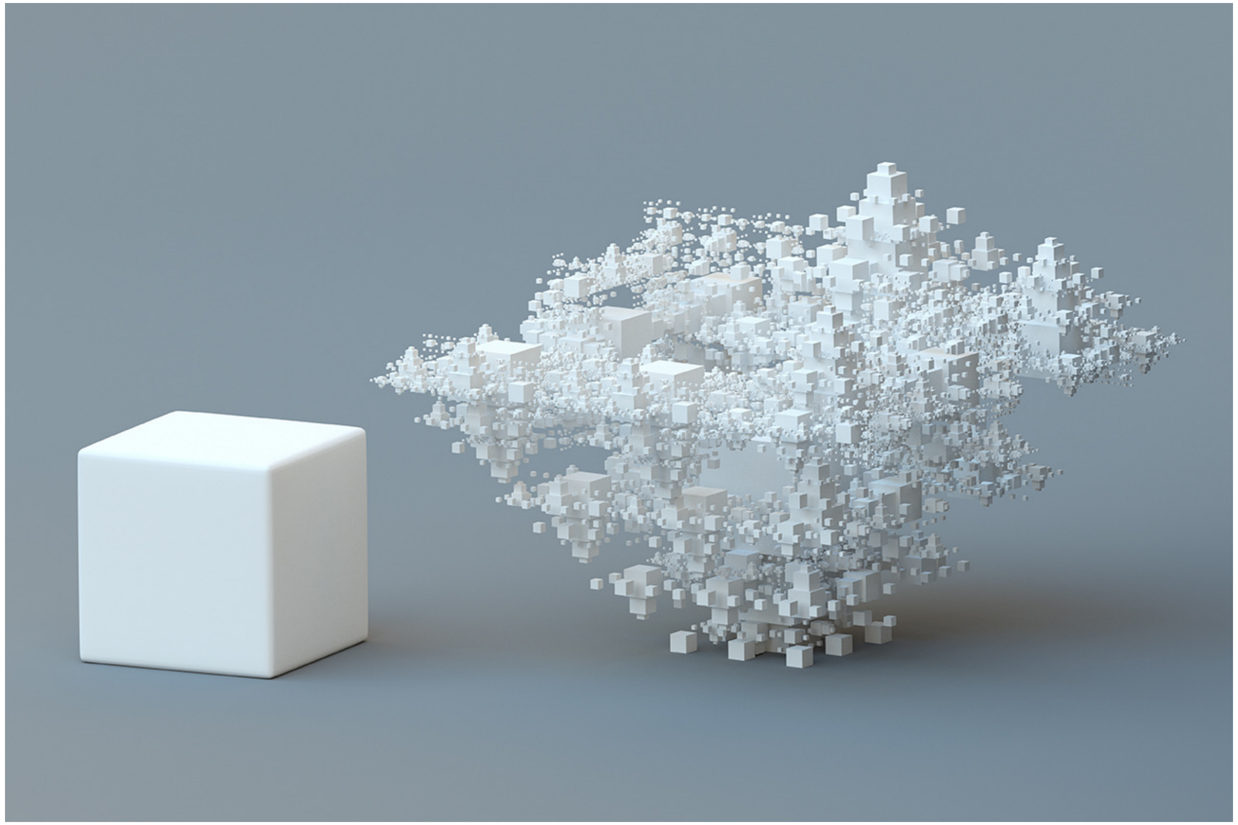
C1 uses C2

C1 is-located-with C2

C1 shares-data-with C2

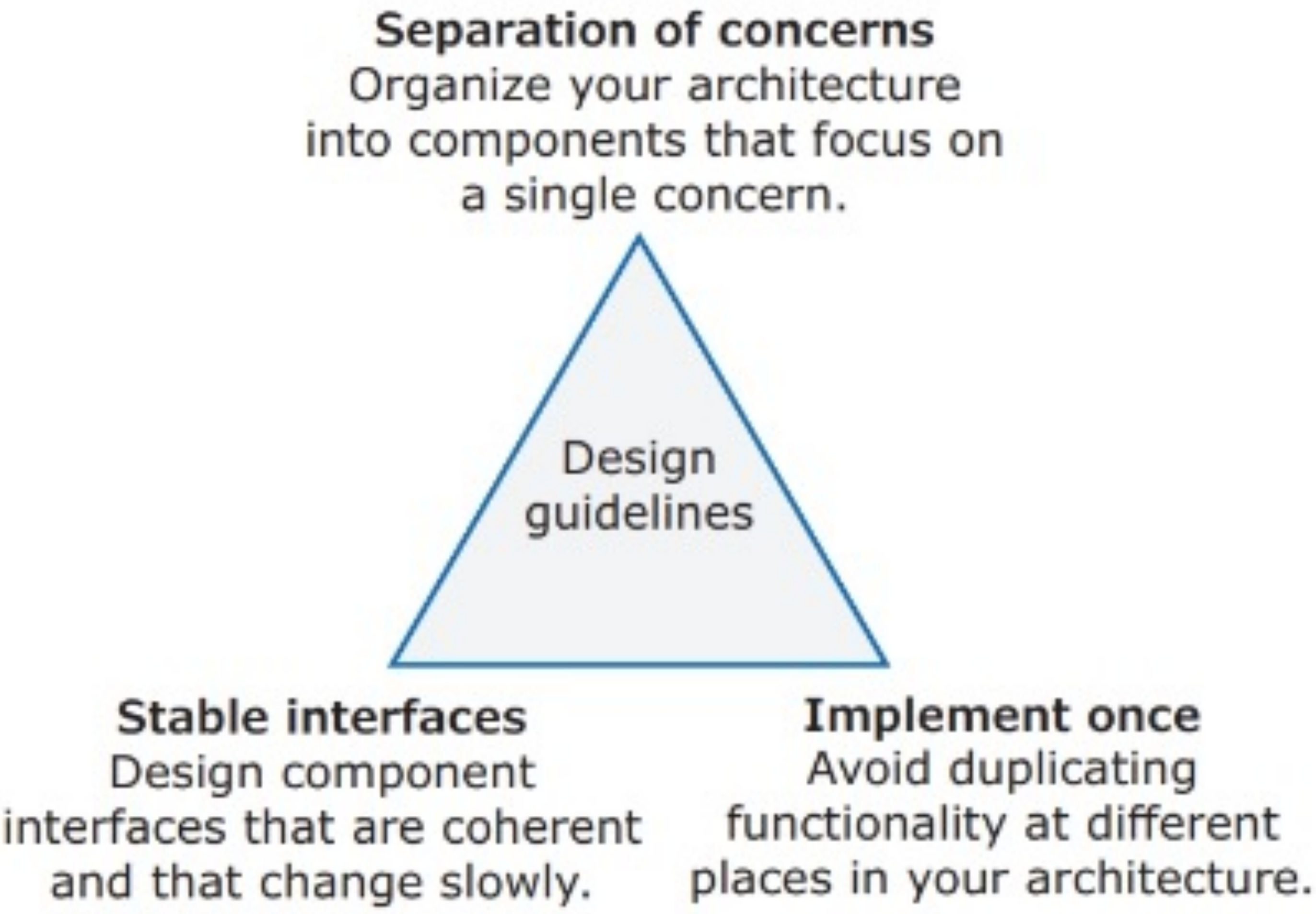
Warning

As the number of components increases, the number of relationships tends to increase at a faster rate



Simplicity is essential (Agile manifesto)

Control complexity



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

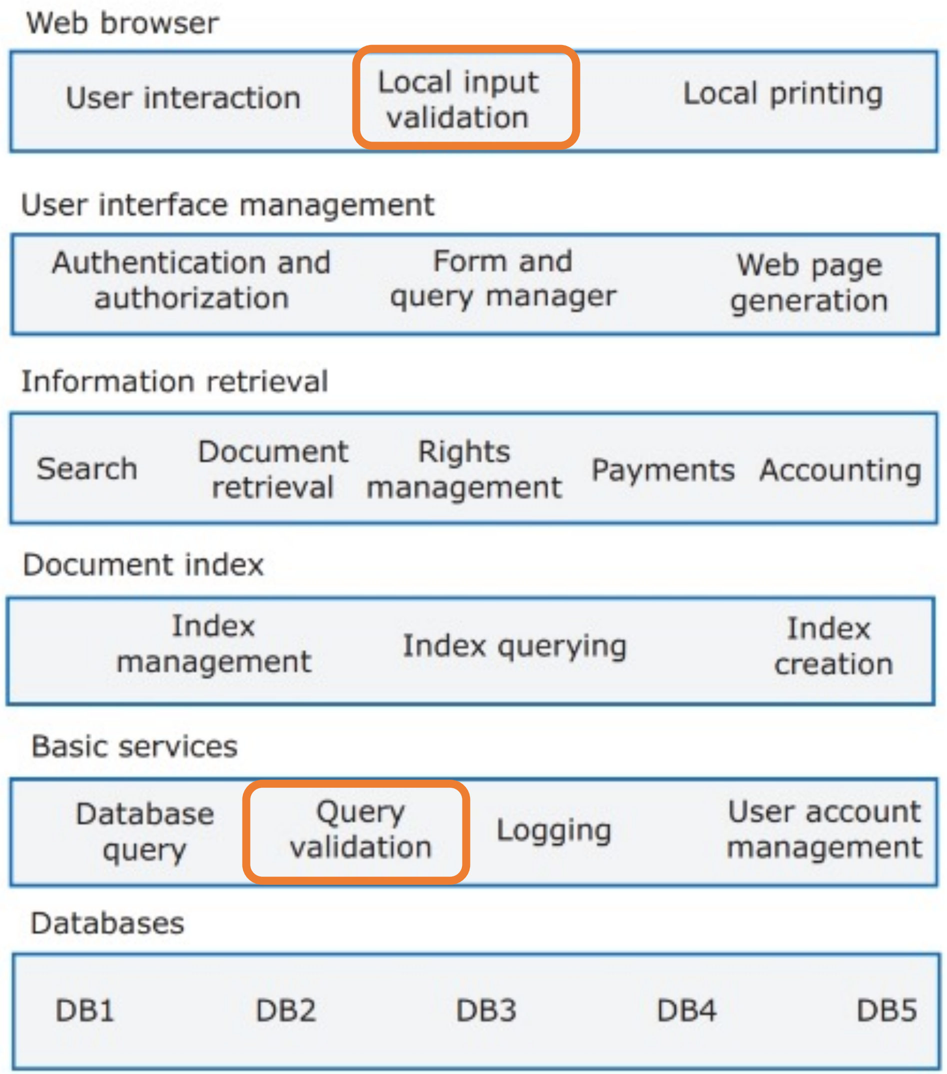
Design guidelines

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

Implement once
Avoid duplicating functionality at different places in your architecture.

Layered architectures

- Each layer is an area of concern and is considered separately from other layers
- Within each layer, the components are independent and do not overlap in functionality
- The architectural model is a high-level model that does not include implementation information



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

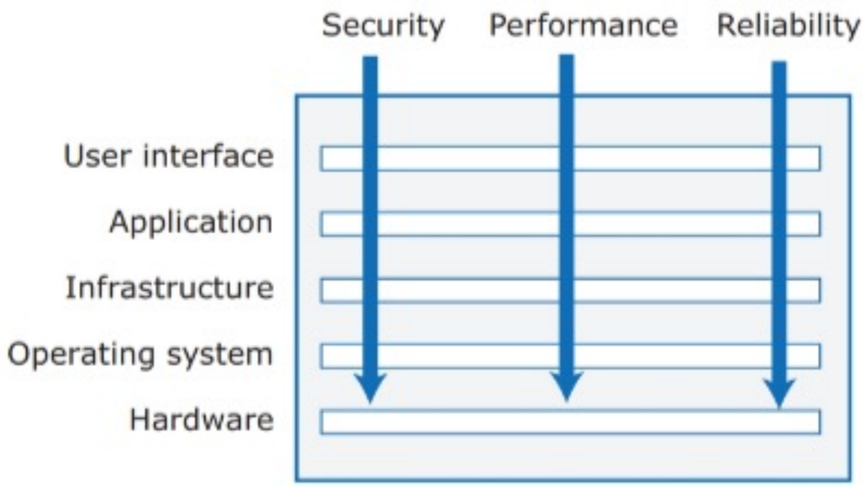
Databases

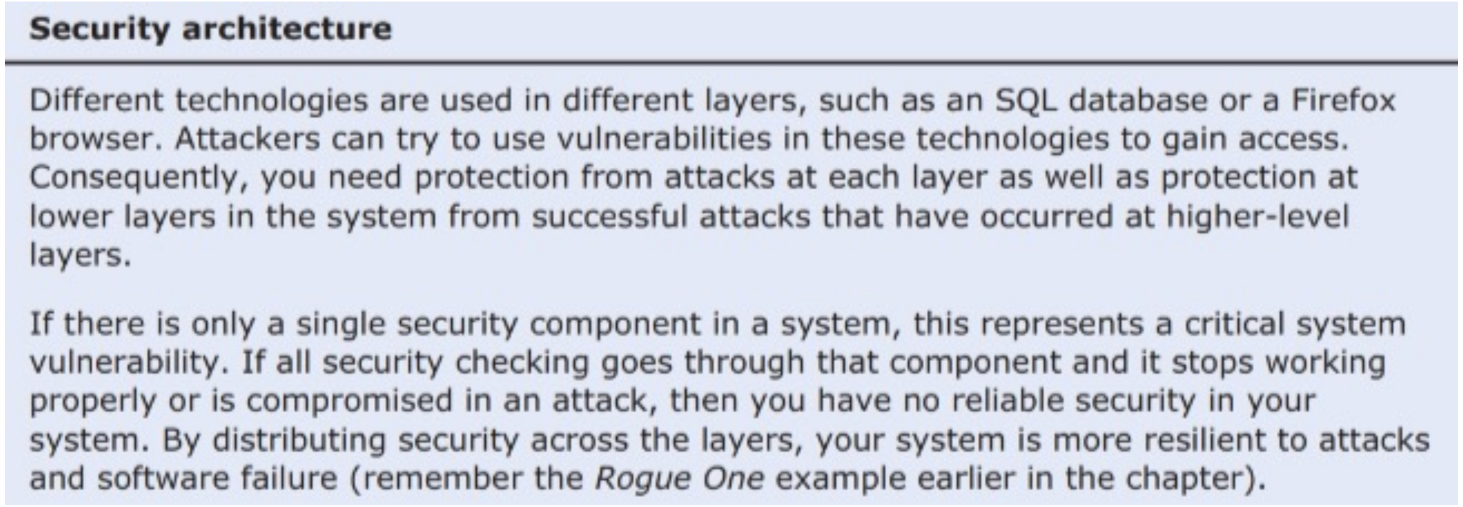
DB1 | DB2 | DB3 | DB4 | DB5

(Concerns may not be always 100% separated in practice)

Cross-cutting concerns

- Systemic concerns affecting whole system
- Every layer must take them into account
- Cross-cutting concerns add interactions between the layers



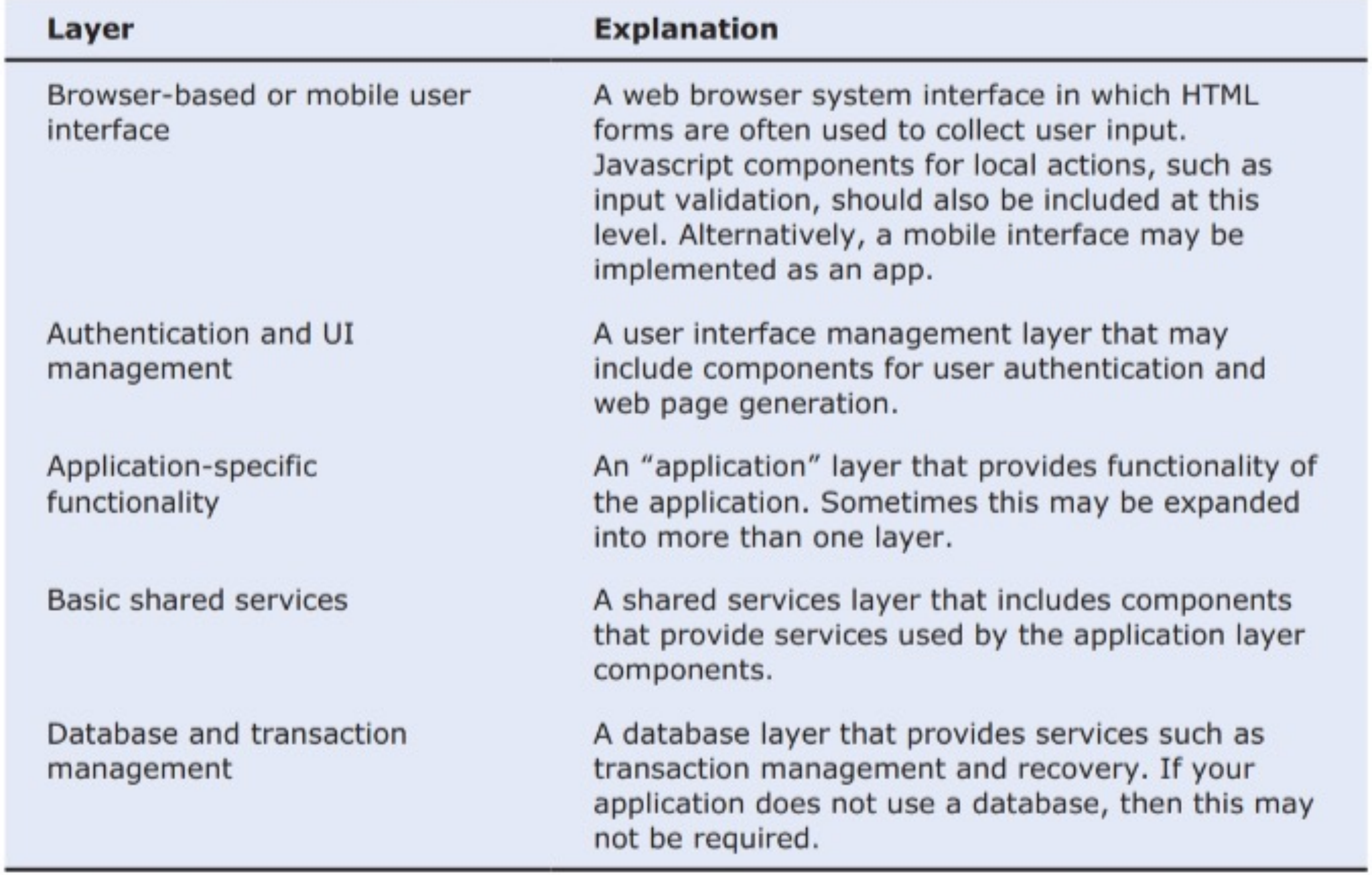


Security architecture

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).

Generic layers for web-based app



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.
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.

Software architect customises this by deciding whether she needs less/more layers

Warning

System decomposition must be done (partly) in conjunction with choosing technologies for your system

- e.g. choice of using relational database affects components at higher layers
- e.g. choice of supporting interfaces on mobile devices calls for using corresponding UI development toolkits

Non-functional quality attributes

System decomposition

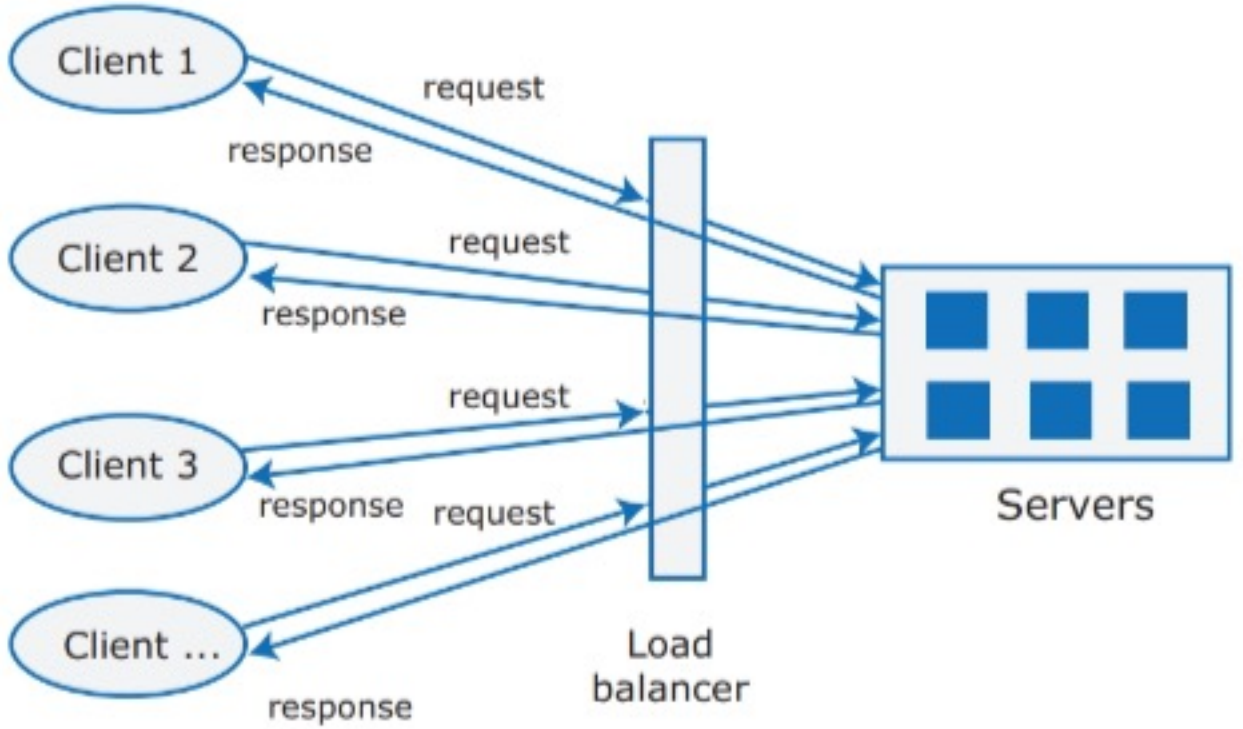
Distribution architecture

Distribution architecture

Define servers and allocation of components to servers

Client-server architecture

Suited to applications in which clients access a shared database and business logic operations on those data

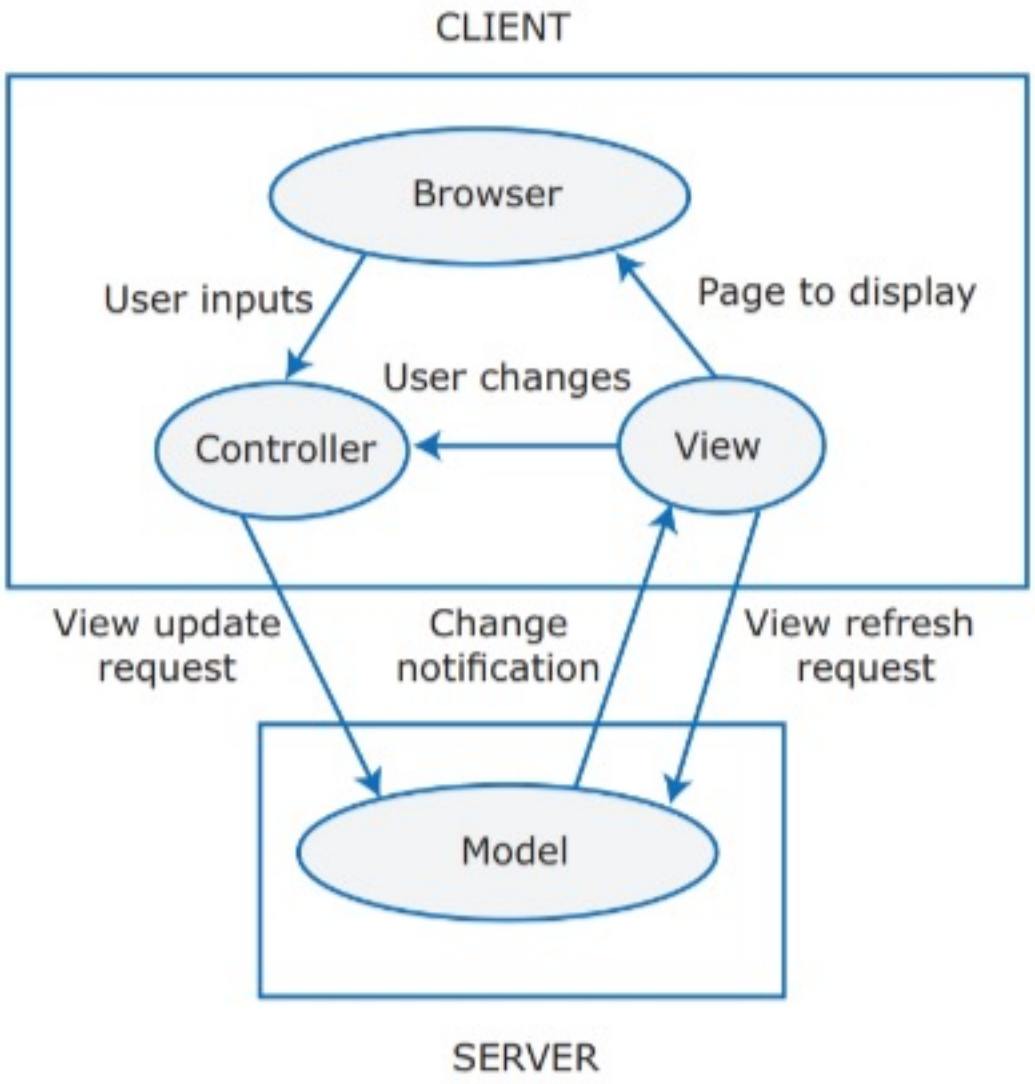


Several servers (e.g. Web servers, db servers), load balanced

Client-server architecture

Model-View-Controller pattern

- Architectural pattern for client-server interaction
- Model decoupled from its presentation
- Each view registers with model so that if model changes all views can be updated



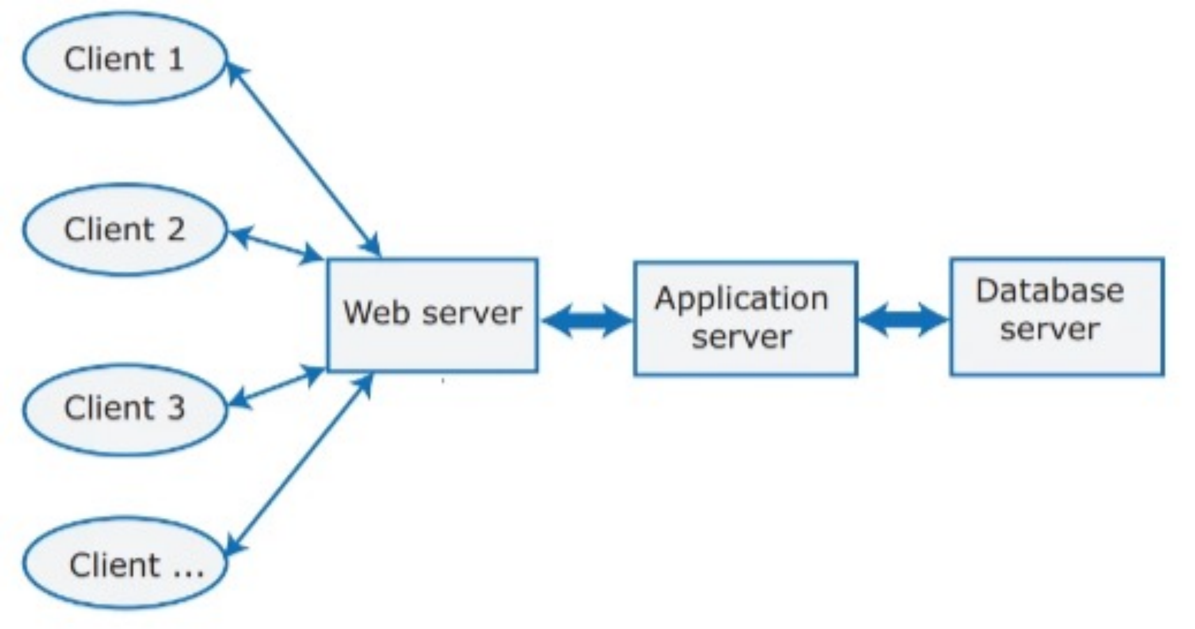
MVC tutorial

https://www.youtube.com/watch?v=p6TIUgZHAT4

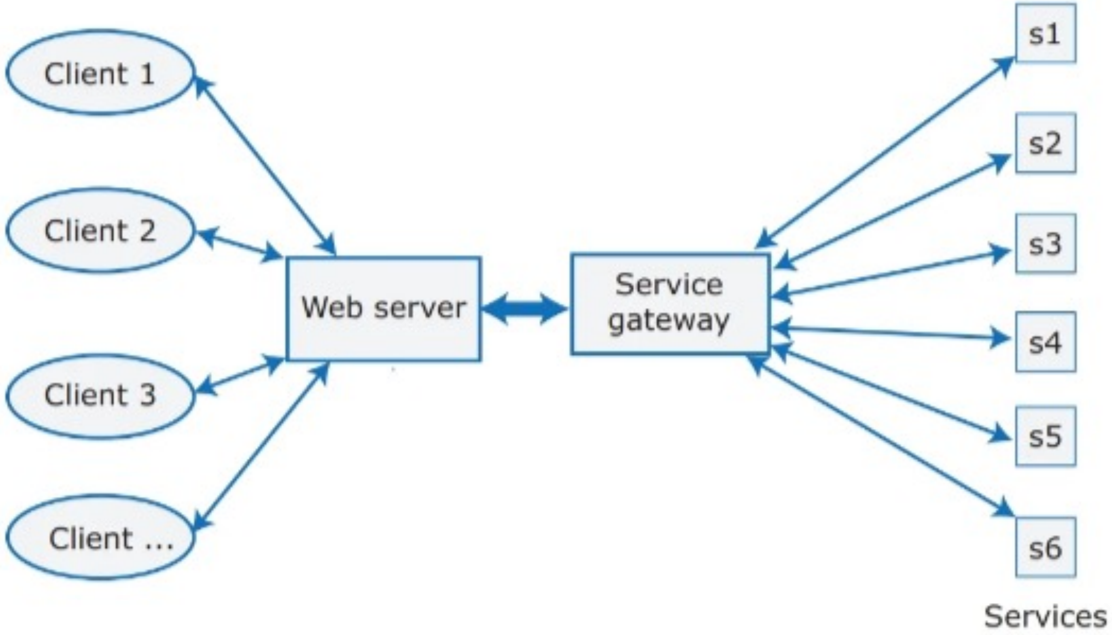
Client-server architecture

Client-server communication usually with HTTP and XML/JSON

Multi-tier web-based architecture



Service-oriented architecture



Choosing distribution architecture

- 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.
- 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.

Non-functional quality attributes

System decomposition

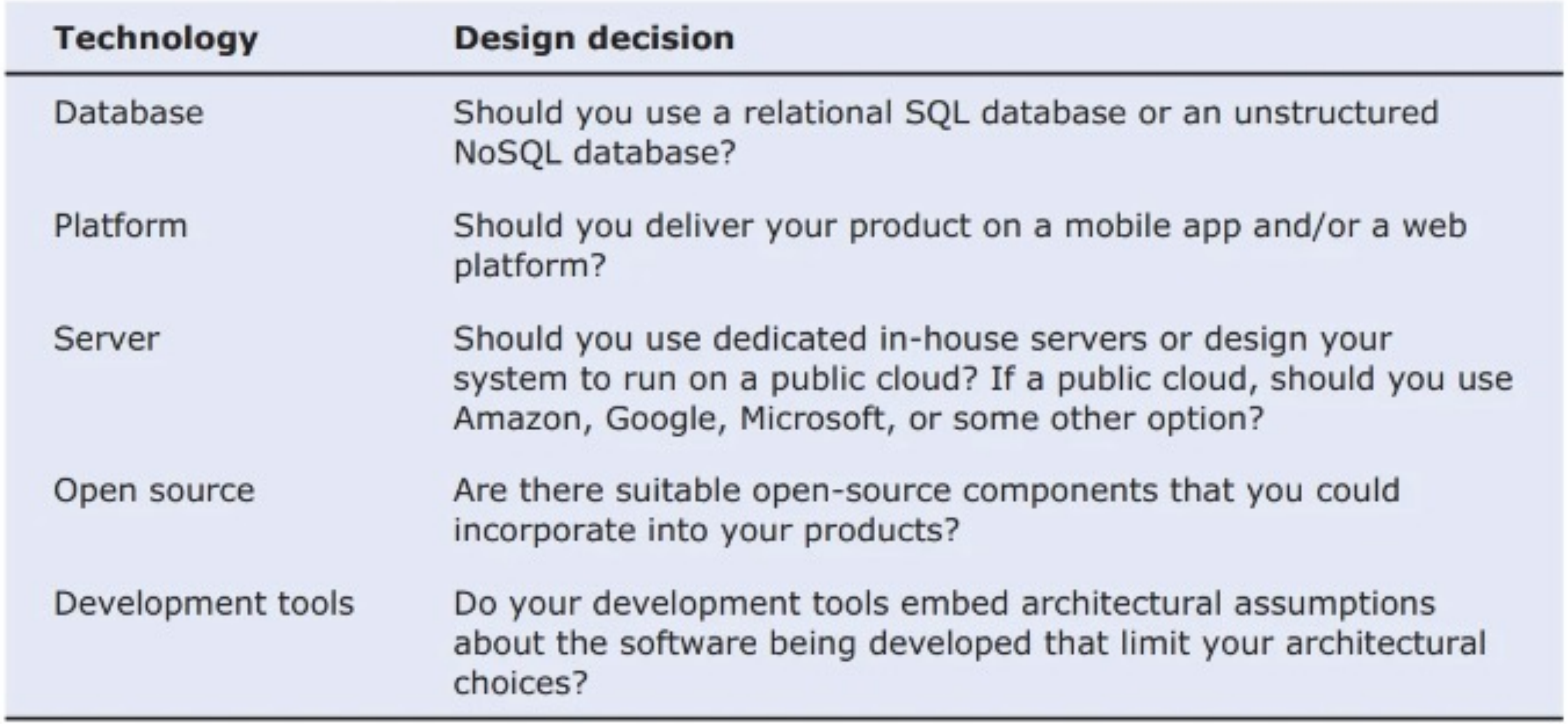
Distribution architecture

Technology issues

Technologies choices

Affect and constrain overall system architecture

Difficult/expensive to change them during development



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

Relational databases

- e.g. MySQL
- data is organised into structured tables
- particularly suitable when
  - you need transaction management and
  - data structures are predictable and simple

NoSQL databases

- e.g. MongoDB
- data has a more flexible, user-defined organization
- more flexible and efficient for data analysis
  - data can be organized hierarchically, efficient concurrent processing of ‘big data’ possible

Why NoSQL databases

https://www.youtube.com/watch?v=0X43QfCfyk0

Delivery platform

- Delivery can be as a web-based or a mobile product
- Mobile issues
  - *Intermittent connectivity*
  - *Processor power*
  - *Power management*
  - *Reduced screen size and on-screen keyboard*

- Different decomposition architectures for the web-based and mobile versions (e.g. to ensure that performance is maintained)

  → Decide early whether to focus on mobile or desktop version of product

Server

- To run on customer servers or to run on the cloud?
  - For consumer products: SOA+cloud
  - For business products, there may be
    - Concerns on cloud security issues
    - Less need to cope with spikes in demand

- If you develop for the cloud, next decision is to choose provider
  - Not easy to move product across cloud providers

Development technology

- Development technologies (e.g. mobile development toolkit, web application framework) influence the architecture of your product
  - e.g. many web development frameworks assume use of model-view-controller architectural pattern

- The development technology that you use may indirectly influence the architecture of your product
  - e.g. if your team is used to relational databases then …

Reference

GLOBAL EDITION
Engineering Software Products
An Introduction to Modern Software Engineering
Ian Sommerville

Chapter 4 – Software architecture