SOFTWARE ENGINEERING
(Week-11)

USAMA MUSHARAF

MS-CS (Software Engineering)

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

AGENDA OF WEEK # 11

- Distributed Software Architecture (Cont…)
  - Microservices
  - REST
- Software Quality Assurance
  - Quality Models

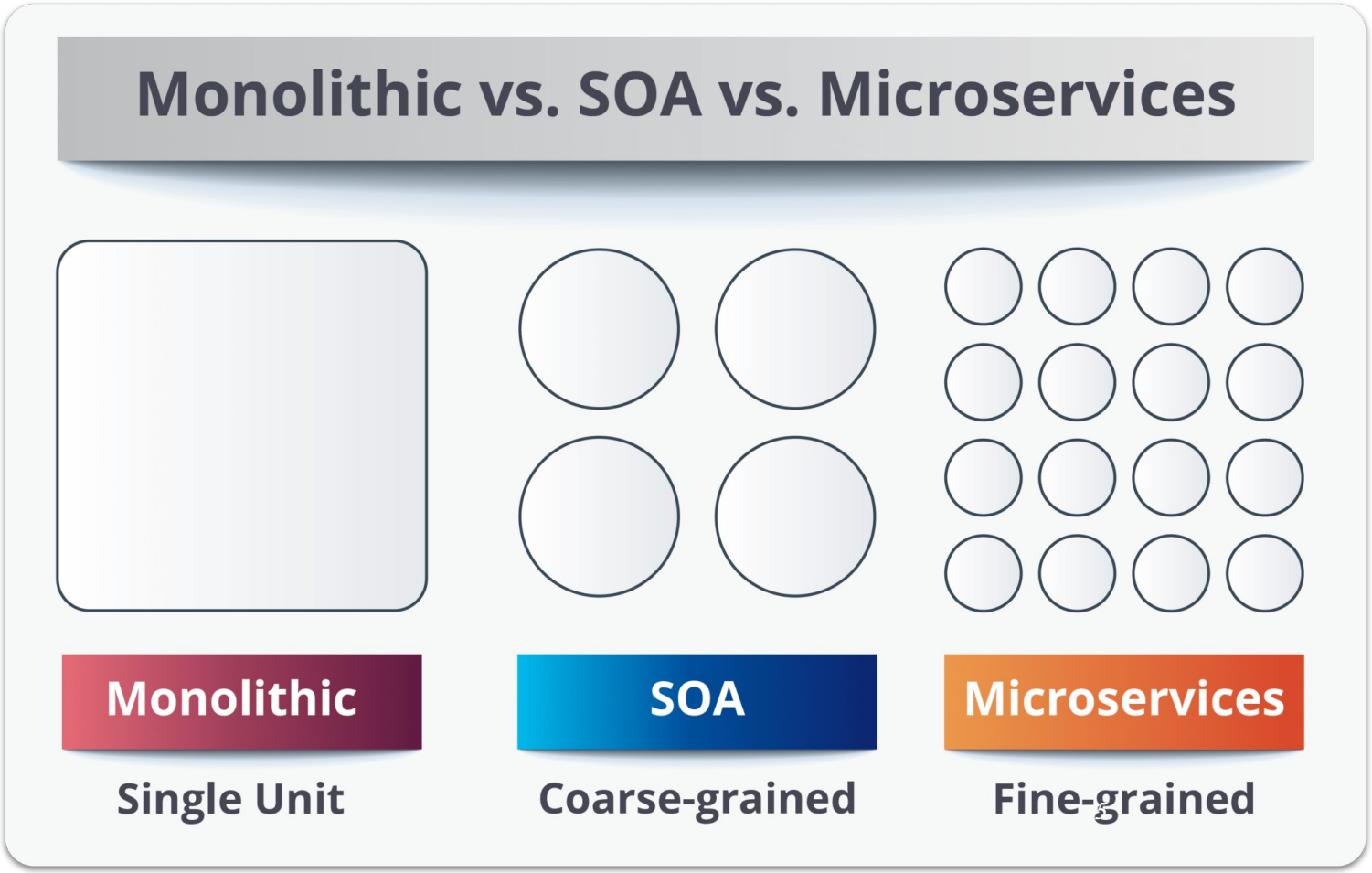
MICROSERVICES

MICROSERVICES

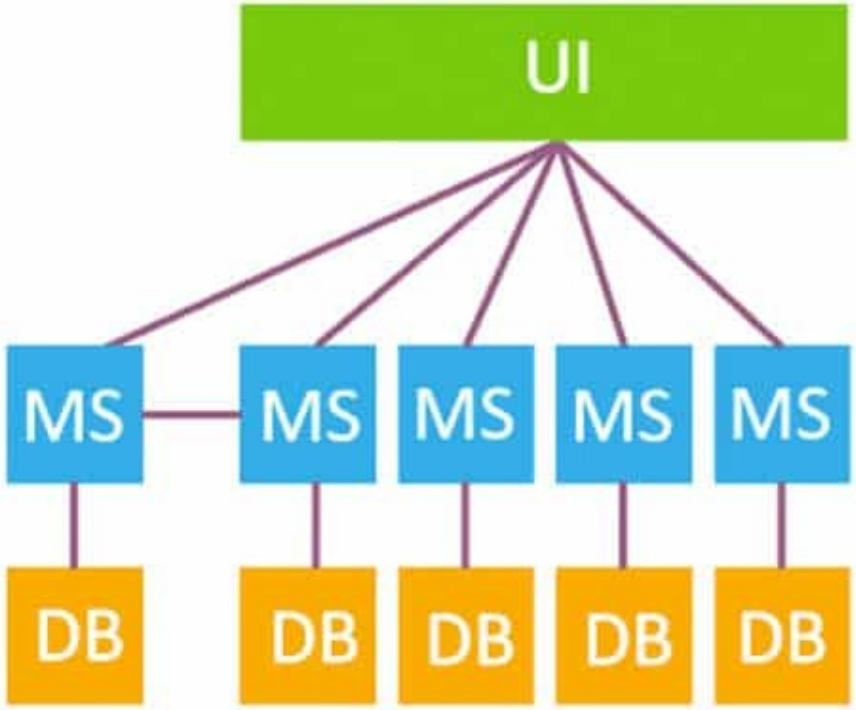
Microservices are small, individually deployable services performing different operations.

Variant of SOA

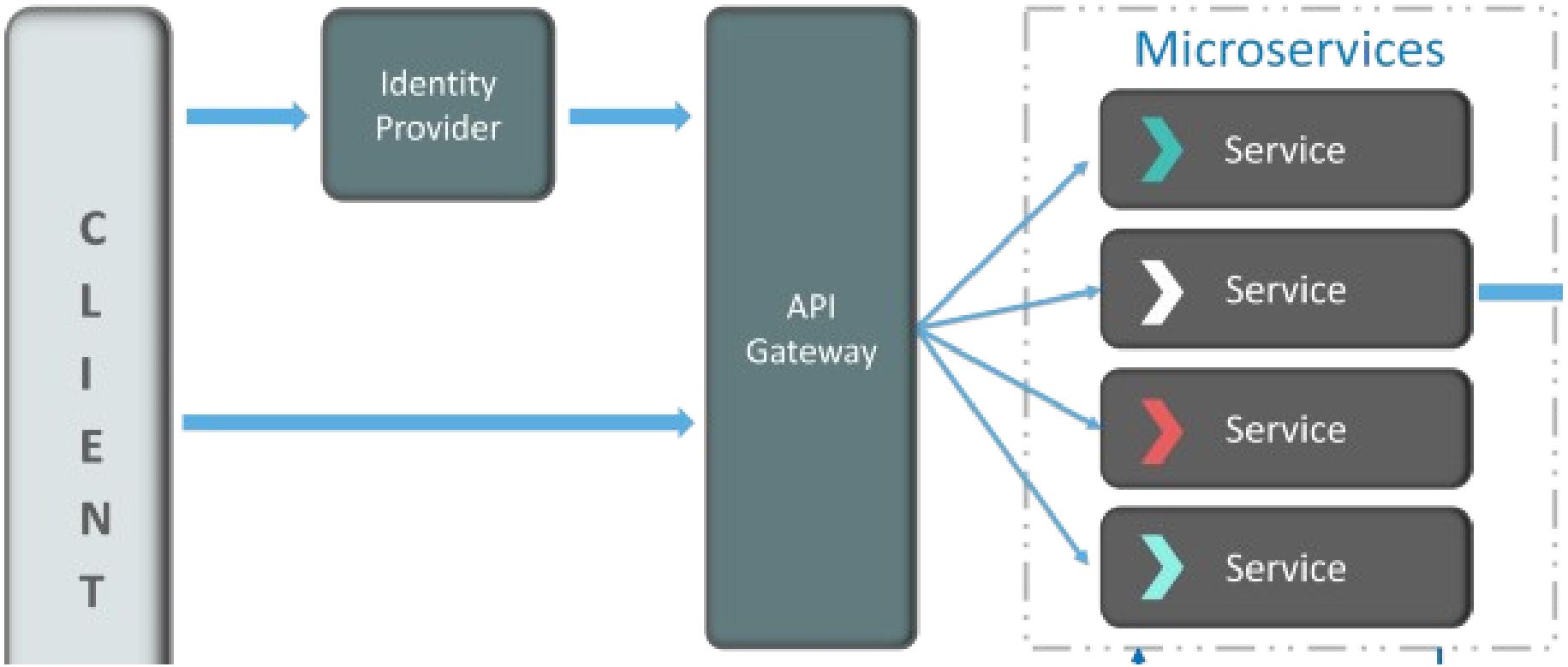
SOA VS MICROSERVICES



SOA VS MICROSERVICES



HOW DOES MICROSERVICES ARCHITECTURE WORK?



COMMUNICATION IN MICROSERVICES

Synchronous Messages:
In the situation where clients wait for the responses from a service, Microservices usually tend to use **REST (Representational State Transfer)** as it relies on a stateless, client-server, and the **HTTP protocol**. This protocol is used as it is a distributed environment each and every functionality is represented with a resource to carry out operations

Asynchronous Messages:
In the situation where clients do not wait for the responses from a service, Microservices usually tend to use protocols such as **AMQP, MQTT**.

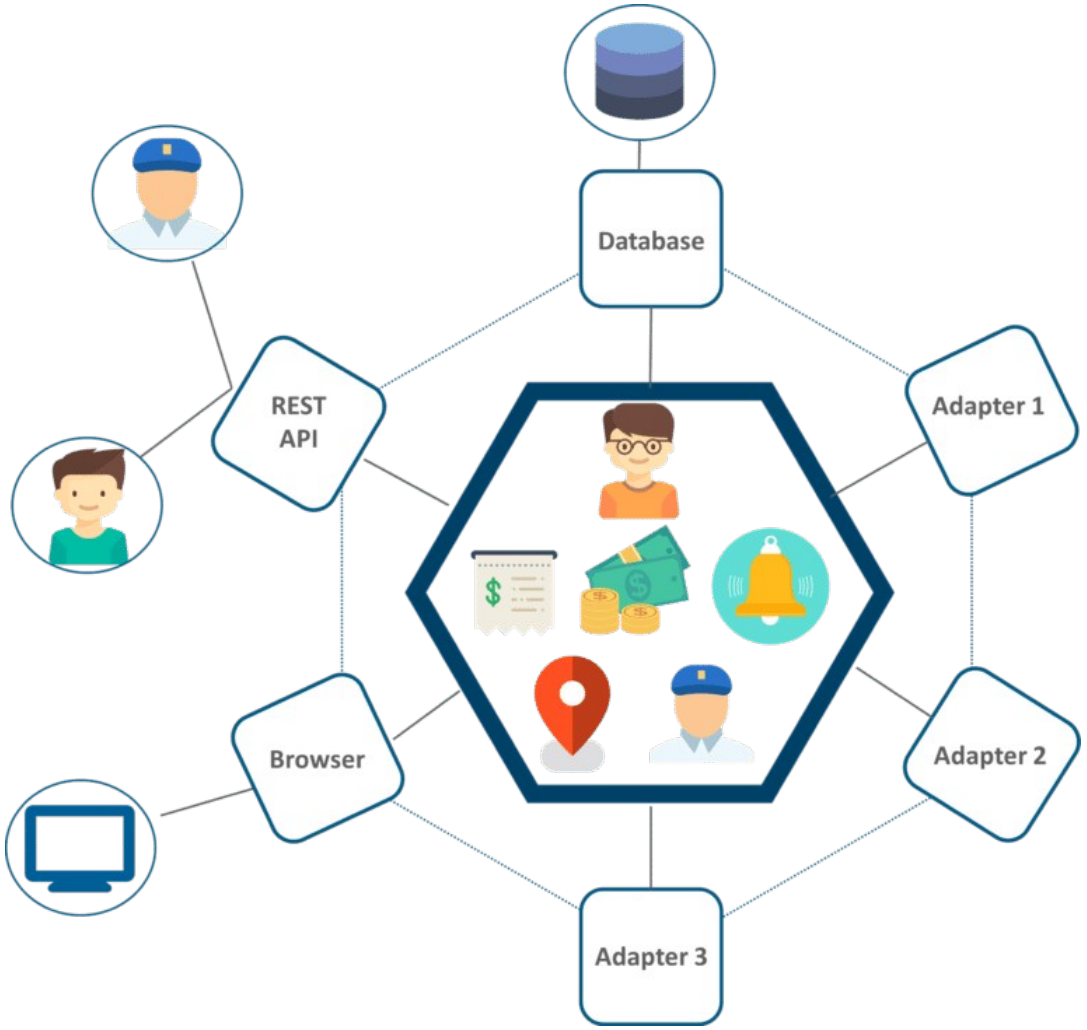
UBER’S PREVIOUS ARCHITECTURE

A REST API is present with which the passenger and driver connect.

Three different adapters are used with API within them, to perform actions such as billing, payments, sending emails/messages that we see when we book a cab.

A MySQL database to store all their data.

All the features such as passenger management, billing, notification features, payments, trip management, and driver management were composed within a single framework.



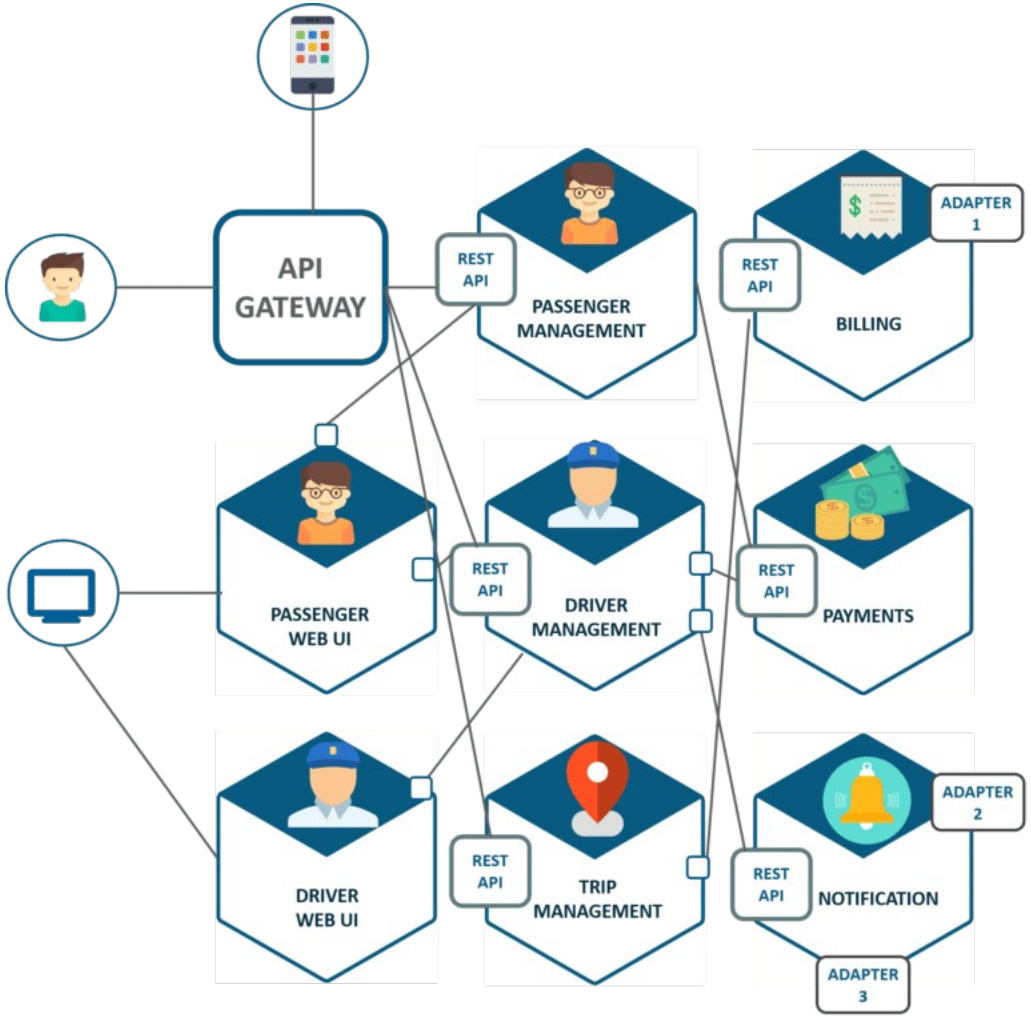
PROBLEM IN UBER'S ARCHITECTURE

All the features had to be re-built, deployed and tested again and again to update a single feature.

Fixing bugs became extremely difficult in a single repository as developers had to change the code again and again.

Scaling the features simultaneously with the introduction of new features was quite tough to be handled together.

SOLUTION IS MICROSERVICES ARCHITECTURE



SOLUTION

The units are individual separate deployable units performing separate functionalities.

For Example: If you want to change anything in the billing Microservices, then you just have to deploy only billing Microservices and don’t have to deploy the others.

All the features were now scaled individually i.e. The interdependency between each and every feature was removed.

REST ARCHITECTURE

REPRESENTATIONAL STATE TRANSFER (REST)

REST, or REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other.

REST is a guideline for building performant and scalable applications.

REPRESENTATIONAL STATE TRANSFER (REST)

Representational State Transfer (REST)

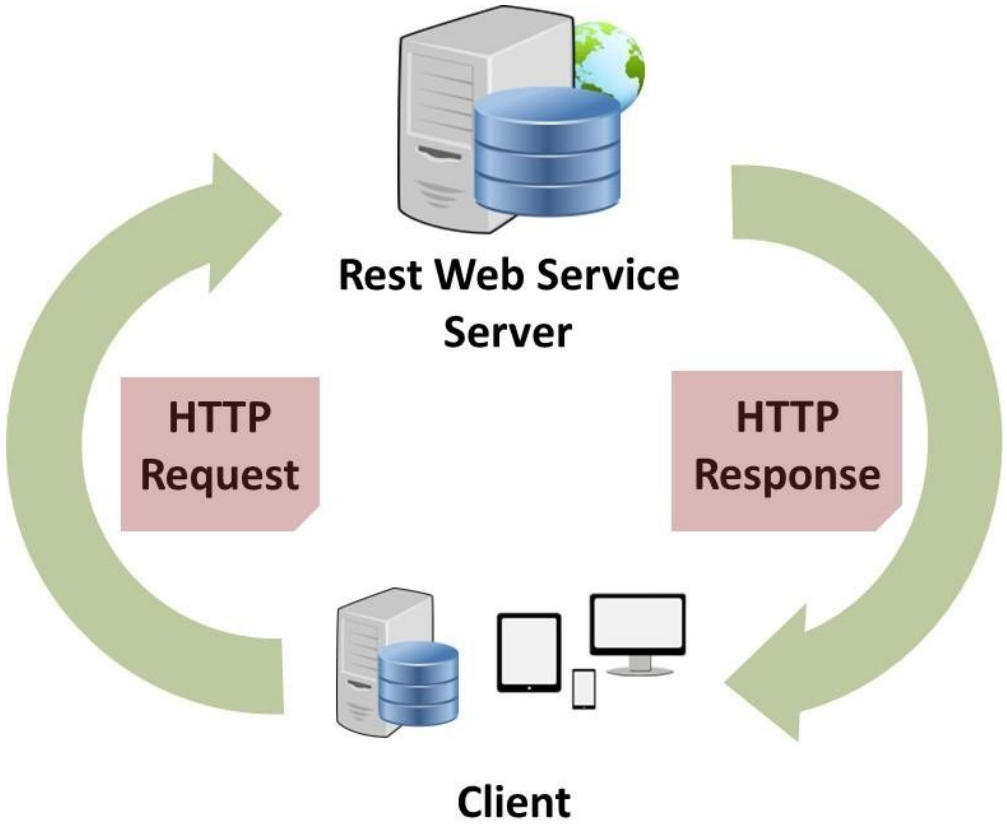
- A style of software architecture for distributed systems such as the World Wide Web.

REST is basically client/server architectural style

- Requests and responses are built around the transfer of "representations" of "resources".

HTTP is the main and the best example of a REST style implementation

- But it should not be confused with REST
- REST is not a protocol
- REST is a guideline



REST PRINCIPLES / ARCHITECTURAL CONSTRAINTS

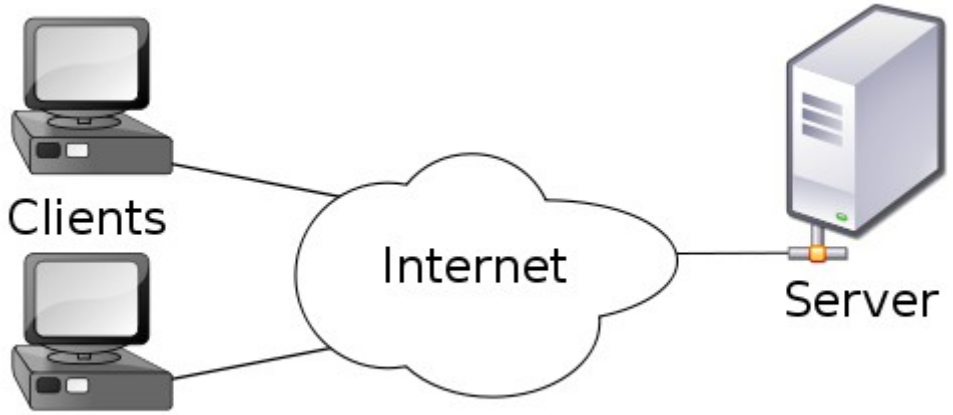
- Client-server
- Stateless
- Cacheable
- Uniform interface
- Layered system
- Code on demand (optional)

1. CLIENT SERVER

Separation of concerns is the principle behind the client-server constraints.

By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.

Client and server can evolve independently.



Client Server Model

2. STATELESS

Statelessness means communication must be stateless in nature as in the client stateless server style,

Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server.

3. CACHEABLE

In order to improve network efficiency, cache constraints are added to the REST style.

Cache constraints require that the data within a response to a request can be cacheable or not

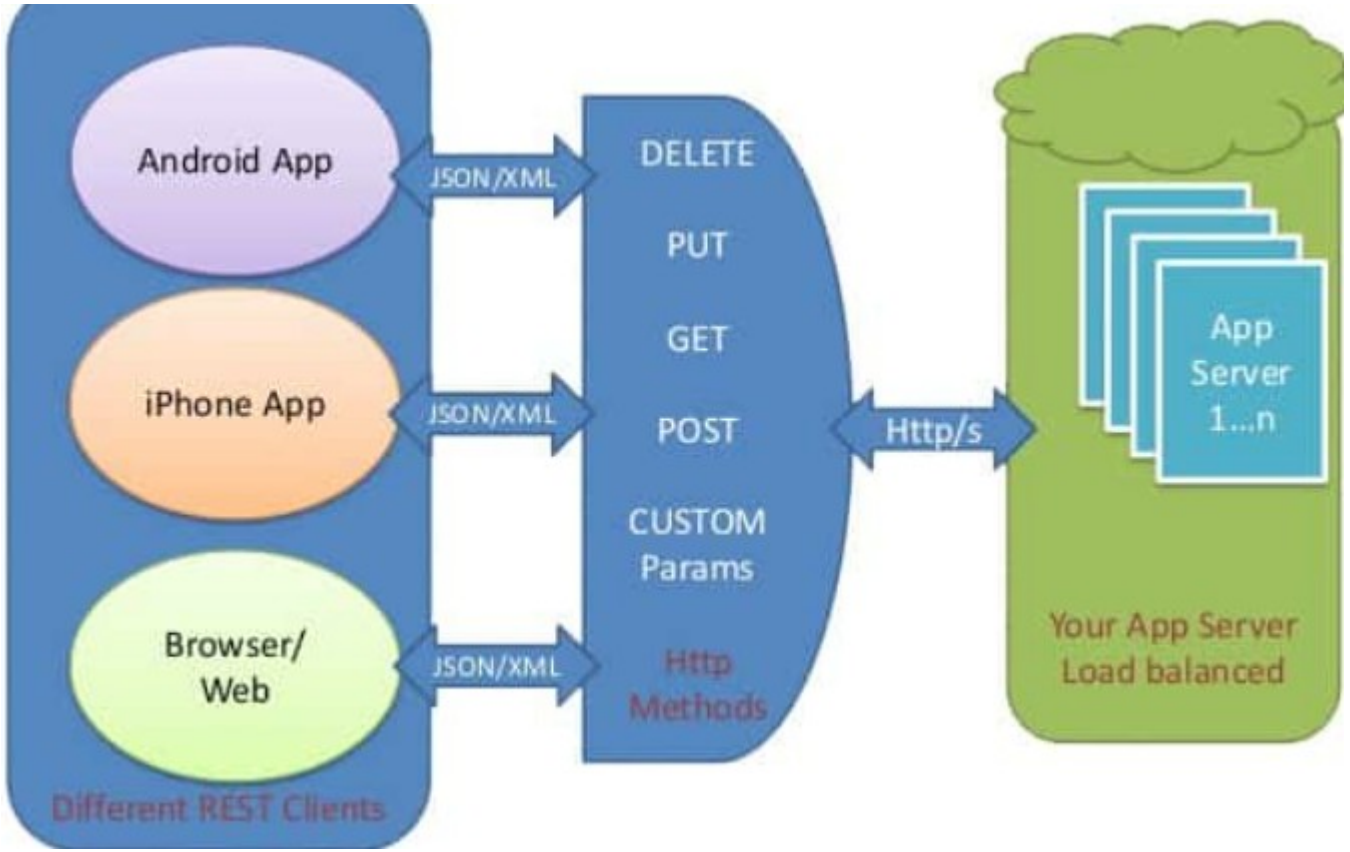
If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

The advantage of adding cache constraints is that they have the potential to partially or completely eliminate some interactions, improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions.

4. UNIFORM INTERFACE

Identification of resources (typically by URI).

Manipulation of resources through representations.



5. LAYERED SYSTEM

The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.

6. CODE ON DEMAND (OPTIONAL)

This states that the server can add more functionality to the REST client, by sending code that can be executable by that client. In the context of the web, one such example is JavaScript code that the server sends to the browser.

For example, a web browser acts like a REST client and the server passes HTML content that the browser renders. At the server side, there is some sort of server-side language which is performing some logical work at the server side. But if we want to add some logic which will work in the browser then we (as server-side developers) will have to send some JavaScript code to the client side and the browser and then execute that JavaScript.

SOFTWARE QUALITY ASSURANCE

QUALITY ATTRIBUTES

Quality attributes depend on each other.

It is impossible to completely satisfy all quality attributes of a software system.

SOFTWARE QUALITY MODELS

Quality is the excellence of the product or service.

From a user's point of view, quality is 'fitness for purpose'.

From the manufacturing point of view, the quality of a product is the conformance to specification.

HIERARCHICAL MODELS

McCall divided software quality attributes into 3 groups.

Each group represents the quality with respect to one aspect of the software system while the attributes in the group contribute to that aspect.

Each quality attribute is defined by a question so that the quality of the software system can be assessed by answering the question.

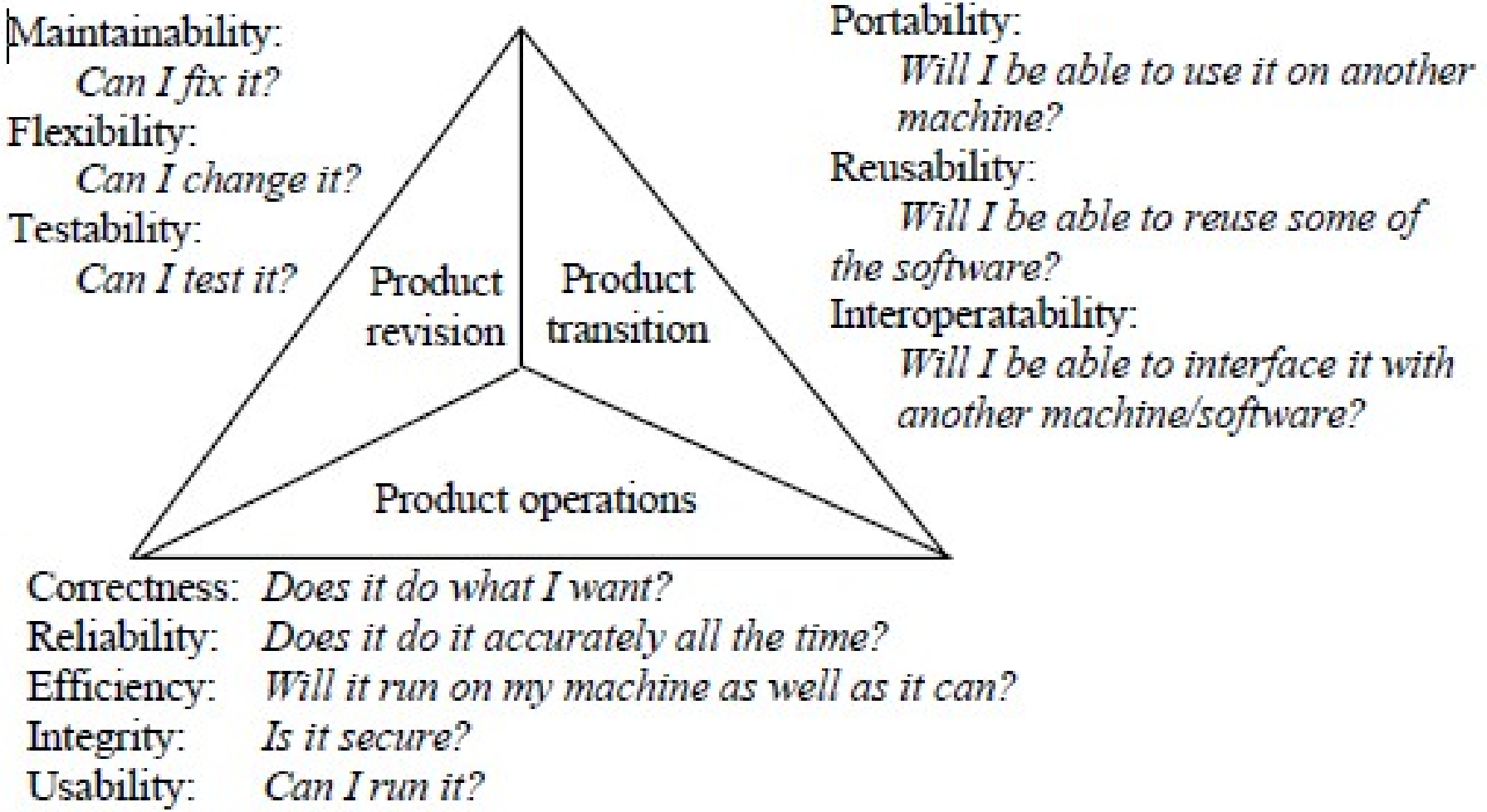


Figure 2.1 McCall's model of software quality

RELATIONAL MODELS

Perry's model contains three types of relationship between the quality attributes.

- The direct relationship
- The inverse relationship
- The neutral relationship

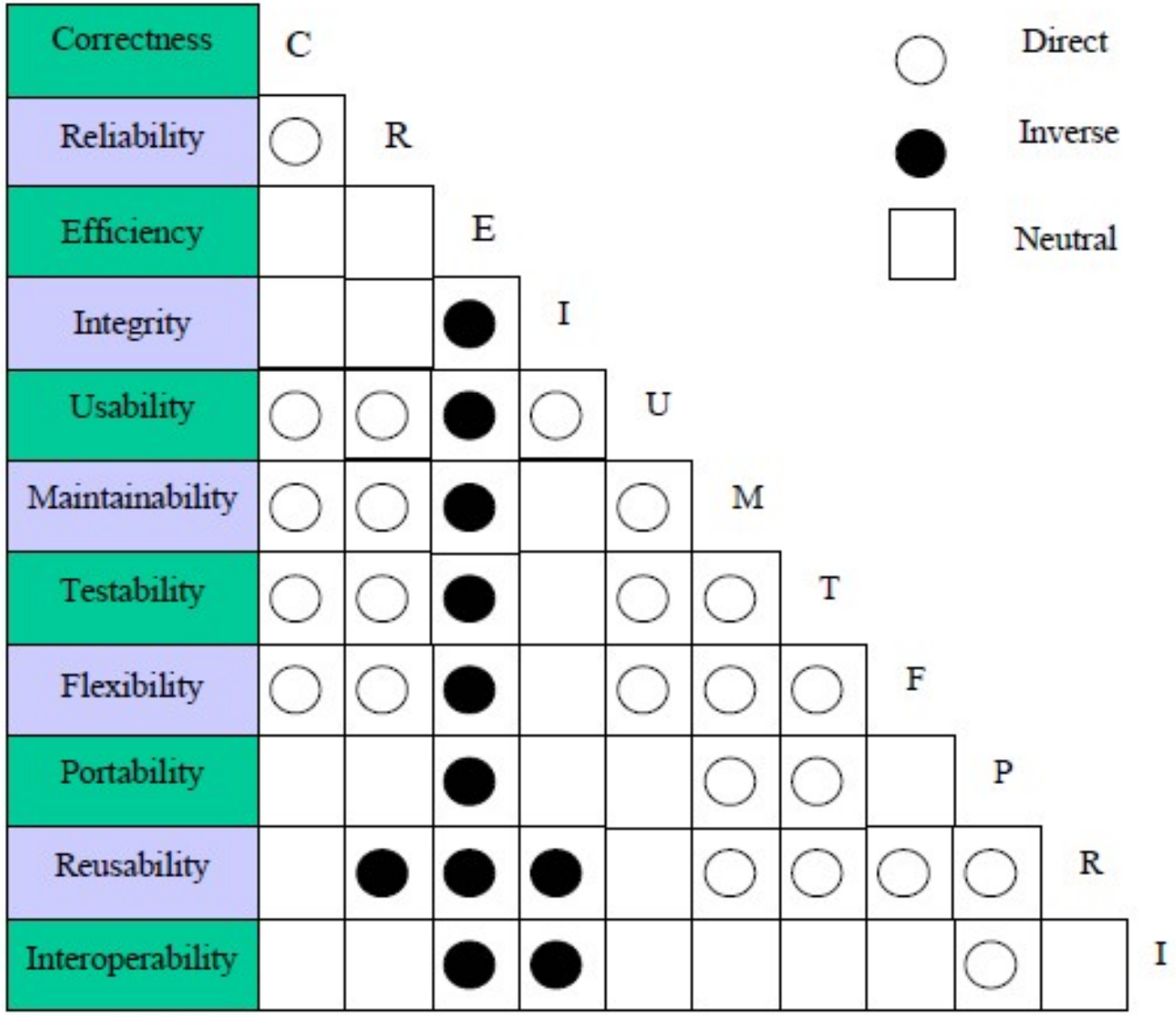


Figure 2.2 Perry's relational model of software quality

RELATIONSHIP B/W QUALITY ATTRIBUTES

Integrity vs. efficiency (inverse):

The control of data access will need additional code, leading to a longer runtime and more storage requirement.

Usability vs. efficiency (inverse):

Improvement of HCI will need more code and data, hence the system will be less efficient.

RELATIONSHIP B/W QUALITY ATTRIBUTES

Maintainability and testability vs. efficiency (inverse):

Compact and optimized code is not easy to maintain and test.

Flexibility, reusability vs. integrity (inverse):

Flexible data structures required for flexible and reusable software increase the data security problem.

Reusability vs. maintainability (direct):

RELATIONSHIP B/W QUALITY ATTRIBUTES

Portability vs. reusability (direct):

Portable code is likely to be easily used in other environments. The code is likely well-structured and easier to be reused.

Correctness vs. efficiency (neutral):

The correctness of code has no relation with its efficiency. Correct code may be efficient or inefficient in operation.

HAVE A GOOD DAY!