QUALITY ATTRIBUTES

SENG 401

WINTER 2019

QUALITY ATTRIBUTE

- A **quality attribute (QA)** is a measureable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders.
- Think of a quality attribute as measuring the **"goodness" of a product** along some dimension of interest to a stakeholder.

QUALITY ATTRIBUTES

- Availability
- Interoperability
- Modifiability
- Performance
- Security
- Testability
- Usability
- …

AVAILABILITY

AVAILABILITY

- **Availability** refers to a property of software that it is there and ready to carry out its task when you need it to be.
- Availability implies **reliability** and **recovery.**
- **Availability** refers to the ability of a system to mask or repair **faults** such that the cumulative service outage period does not exceed a required value over a specified time interval.

AVAILABILITY

- Availability is closely related to **security**. A denial-of-service attack is explicitly designed to make a system fail (unavailable).
- Availability is also closely related to **performance**, because it may be difficult to tell when a system has failed and when it is simply being outrageously slow to respond.
- Availability is closely allied with **safety**, which is concerned with keeping the system from entering a hazardous state.

AVAILABILITY

- Availability is about minimizing service outage time by mitigating **faults**.
- One of the most demanding tasks in building a high-availability, fault-tolerant system is to understand the nature of the **failures** that can arise during operation.
- A **failure** occurs when the system no longer delivers a service that is consistent with its specification. A **fault** has the potential to cause a failure.

AVAILABILITY

- The availability of a system can be calculated as the probability that it will provide the specified services within required bounds over a specified time interval:

$$\frac{MTBF}{(MTBF + MTTR)}$$

  - MTBF: mean time between failures
  - MTTR: mean time to repair

AVAILABILITY

- The availability provided by a computer system or hosting service is frequently expressed as a **Service Level Agreement** (SLA)

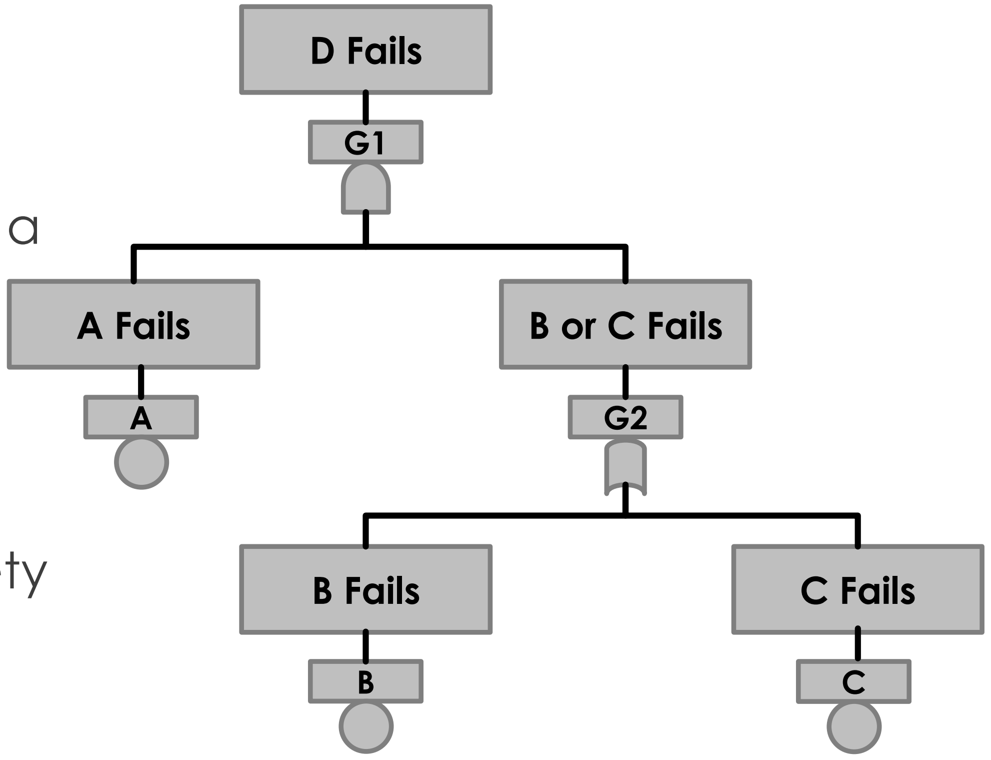
System Availability Requirements		
Availability	**Downtime/90 Days**	**Downtime/Year**
99.0%	21 hours, 36 mins	3 days, 15.6 hours
99.9%	2 hours, 10 mins	8 hours, 46 secs
99.99%	12 mins, 58 secs	52 mins, 34 secs
99.999%	1 min, 18 secs	5 mins, 15 secs
99.9999%	8 secs	32 secs

FAILURE

- Failure is not only an option, it’s almost **inevitable.**
- What will make your system safe and available is planning for the occurrence of failure(s):
  - Understand type of failures your system is prone to
  - Understand consequences of each failure

FAILURE

- Techniques to handle failure:
  - **Hazard Analysis:** a technique to catalog the hazards that can occur during the operation of a system according to its severity (Catastrophic, Hazardous, Major, Minor, No effect)
  - **Fault Tree Analysis:** a technique that specifies a state of the system that negatively impacts safety or reliability, and then analyzes the system's context and operation to find all the ways the undesired state could occur

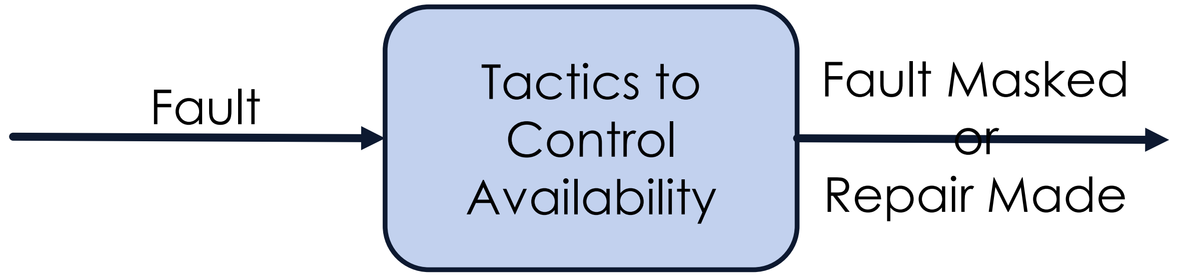


D fails if A fails AND either B OR C fails

TACTICS FOR AVAILABILITY

- A **failure** occurs when the system no longer delivers a service that is consistent with its specification. A **fault** has the potential to cause a failure.

- **Availability tactics** are designed to enable a system to endure faults so service remains compliant with its specification. Tactics keep faults from becoming failures. Categories:
  - Fault Detection
  - Fault Recovery
  - Fault Prevention



INTEROPERABILITY

INTEROPERABILITY

- **Interoperability** is about the degree to which two or more systems can usefully ***exchange meaningful information*** via ***interfaces*** in a particular context.
    - **Syntactic Interoperability:** the ability to exchange data
    - **Semantic Interoperability:** the ability to correctly interpret the data being exchanged

INTEROPERABILITY

- Critical Concepts:
  - Exchange Information
  - Interface
- **API:** a syntactic description of a component’s programs and the type and number of their parameters.

INTEROPERABILITY

- Why Interoperability:
  - You **provide** a service to other systems
  - You **consume** services from other systems
- Aspects of Interoperability:
  - **Discovery:** The consumer of service must discover the location, identity, and the interface of the service
  - **Handling the response**

INTEROPERABILITY

- Interoperability technologies on the Web:
  - **SOAP** (Simple Object Access Protocol)
  - **REST** (Representation State Transfer)

INTEROPERABILITY

REST

- Offers **simplicity**
- Semantic interoperability NOT guaranteed
- **Self-descriptive**
- **Stateless** protocol
- High **performance**
- Appropriate for read-only functionality
- Appropriate for **mashups**

Request Example in REST:

http://www.XYZdirectory.com/phonebook/Userinfo/99999

SOAP

- Offers **completeness**
- Standardized interoperability
- Big size of individual messages
- Structured messages
- High **security**
- High **availability**

Request Example in SOAP:

```
< ? xml version= "l.O" ? >
 <soap : Envelope xmlns:soap=http://www.w3.org/2001/12/soap-
envelope
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding " >
   < soap:Body pb="http://www.XYZdirectory.com/phonebook " >
     <pb : GetUserinfo>
       <pb : Useridentifier> 99999 < /pb : Useridentifier>
     </pb : GetUserinfo>
   < / soap : Body>
 < / soap : Envelope>
```

MODIFIABILITY

MODIFIABILITY

- The software change is not only **constant** but **ubiquitous**.
- Reason for change:
  - Add new feature
  - Retire old features
  - Fix defects
  - Tighten security
  - Improve performance
  - Enhance user experience
  - Embrace new technology, new platform, new protocol, new standard
  - Make systems work together

MODIFIABILITY

- **Modifiability** is about change, its cost and risk.
- Architectural considerations of modifiability:
  - **What can change?** (e.g. function, platform, environment, protocol, …)
  - **What is the likelihood of the change?**
  - **When is the change made and who makes it?** (e.g. implementation modifying source code, compile using compile-time switches, build by choice of libraries, configuration setting by parameter setting, execution by plugins and parameter setting)
  - **What is the cost of change?** (e.g. the cost of introducing the mechanism and the cost of making modifications)

MODIFIABILITY

- **Coupling:** the probability that a modification to one module will propagate to other modules.
  - High coupling is an enemy of modifiability
- **Cohesion:** how strongly the responsibilities of a module are related.
  - High cohesion is good

PERFORMANCE

PERFORMANCE

- **Performance** is about time and the software system’s ability to meet timing requirements.
- Performance used to be the first and most important quality attribute.
- Performance is linked with **scalability**: increasing the capacity of the system to work while performing well.
- Performance is also related to **concurrency**: operations occurring in parallel.

PERFORMANCE GENERAL SCENARIO

- A Performance scenario begins with an **event** arriving at the system.
- **Responding** correctly to the event requires resources to by consumed.

PERFORMANCE GENERAL SCENARIO

- Events can arrive in:
  - Predictable patterns (**Periodic**: at regular time intervals)
  - Mathematical distribution (**Stochastic**: according to some probabilistic distribution)
  - Unpredictable (**Sporadic**: neither periodic nor stochastic)

PERFORMANCE GENERAL SCENARIO

- The response of the system to a stimulus can be measured by:
  - **Latency**: time between the arrival of stimulus and the system's response
  - **Deadlines in processing**
  - **Throughput:** the number of transactions the system can process in a unit of time
  - **Jitter:** allowable variation in latency
  - **Number of events not processed**

SECURITY

SECURITY

- **Security** is a measure of the system's ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized.

- Security characteristics (CIA):
  - **Confidentiality** is the property that data or services are protected from unauthorized access. (e.g. hackers cannot access your income tax)
  - **Integrity** is the property that data or services are not subject to unauthorized manipulation. (e.g. values cannot be changed by other staff)
  - **Availability** is the property that the system will be available for legitimate use. (e.g. you can order book from an online bookstore)

SECURITY

- Other Security characteristics:
  - **Authentication** verifies the identities of the parties to a transaction. (e.g. Authenticating if an email is from a bank)
  - **Nonrepudiation** guarantees that the sender of a message cannot deny having sent the message, and the recipient cannot deny having received it.
  - **Authorization** grants a user the privileges to perform a task. (e.g. you can have access to your bank account and other people cannot)

SECURITY

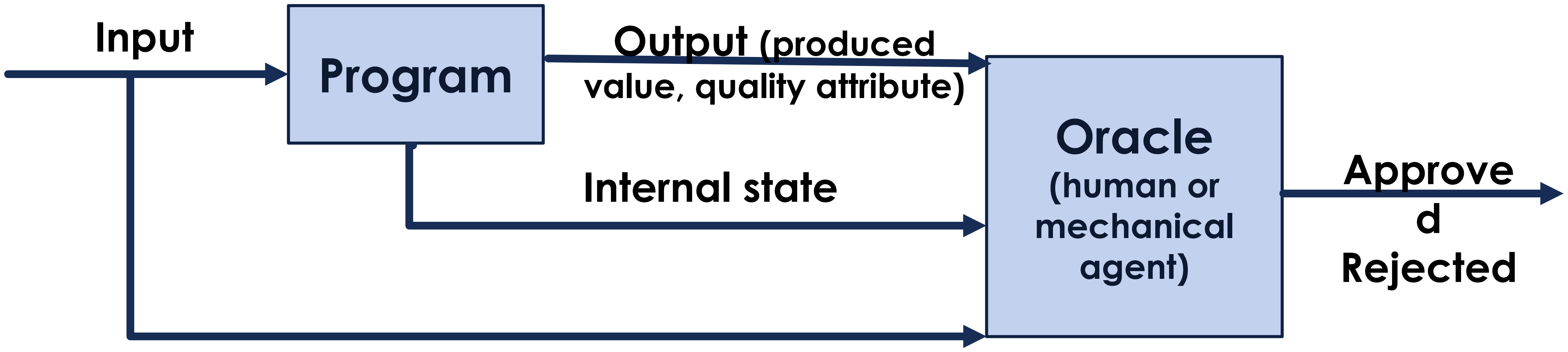
- An **Attack** is an attempt to break CIA.
- Approaches to achieving security:
  - Detect attacks (e.g. security checkpoints)
  - Resist attacks (e.g. armed guards)
  - React to attacks (e.g. automatic lock)
  - Recover from successful attacks (e.g. off-site backup)

TESTABILITY

TESTABILITY

- Industry estimates indicate that between 30 and 50 percent of the cost of developing well-engineered systems is taken up by **testing**.
- Software **testability** refers to the ***ease*** with which software can be made to demonstrate its faults through testing.
- Testability refers to the **probability** that it will fail on its next test execution.

A MODEL OF TESTING


Input
Program
Output (produced value, quality attribute)
Internal state
Oracle (human or mechanical agent)
Approved
Rejected

TESTABILITY: EXAMPLE

- **Chaos Monkey** used by Netflix: a tool invented in 2011 by Netflix to test the resilience of its IT infrastructure. It works by intentionally disabling computers in Netflix's production network to test how remaining systems respond to the outage.

- **Chaos Engineering**: is the discipline of experimenting on a software system in production in order to build confidence in the system’s capability to withstand turbulent and unexpected conditions.

TESTABILITY

- Tactics for testability:
  - Adding **controllability** and **observability** to the system. Specific tactics:
    - Specialized interfaces (e.g. set and get methods, report method, reset method, …)
    - Record/playback
    - Localize state storage
    - Abstract data sources (control input)
    - Sandbox: isolating an instance of the system to enable experimentation
    - Executable assertions (specific data declarations)
  - Limiting complexity in the system’s design
    - Limit structural complexity (resolve, isolate or encapsulate dependencies)
    - Limit nondeterminism (limiting behavioral complexity: e.g. unconstrained parallelism)

USABILITY

USABILITY

- **Usability** is concerned with how ***easy*** it is for the user to ***accomplish*** a desired task and the kind of user support the system provides.
  - Learning system features (How to make the learning of system easier?)
  - Using a system efficiently (How to make the user more efficient in operation?)
  - Minimizing the impact of errors
  - Adapting the system to the user needs (e.g. fill forms based on past entries)
  - Increasing confidence and satisfaction (e.g. provide feedback)

TACTICS FOR USABILITY

- Separate the User Interface
  - Model-View-Controller (MVC) is the dominant separation pattern
  - MVC makes it easy to provide multiple views of the data, supporting user initiative
- Support User initiative (e.g. Cancel, Undo, Pause/Resume, Aggregate)
- Support System initiative
  - Maintain task model (e.g. knowing the sentences start with capital letter)
  - Maintain user model (e.g. user can customize the interface)
  - Maintain system model (e.g. progress bar that predicts the completion time)

OTHER QUALITY ATTRIBUTES

- **Variability:** the ability of a core asset to adapt to usages in the different product contexts that are within the product line scope. (special form of modifiability)
- **Portability:** the ease with which software that was built to run on one platform can be changed to run on a different platform. Portability is achieved by:
  - Minimizing platform dependencies in the software
  - Isolating dependencies to well-identified locations
  - Writing the software to run on a VM (e.g. Java Virtual Machine) that encapsulates all the platform dependencies within

OTHER QUALITY ATTRIBUTES

- **Development Distributability:** the quality of designing the software to support distributed software development
- **Scalability and elasticity:**
  - Horizontal Scalability: adding more resources to logical units (e.g. adding another server to a cluster of servers)
  - Vertical Scalability: adding more resources to a physical unit (e.g. adding more memory to a single computer)
- The problem with scaling is how to effectively utilize the additional resources.
- In cloud environments, horizontal scalability is called **elasticity**.
- Scalability scenarios deal with the impact of **adding** or **removing** resources, and the measures will reflect associated **availability** and the **load** assigned to existing and new resources.

OTHER QUALITY ATTRIBUTES

- **Mobility:** deals with the problems of movement and affordances of a platform. (e.g. size and type of display, type of input devices, availability and volume of bandwidth, battery life)
- **Monitorability:** the ability of the operations staff to monitor the system while it is executing.

USE CASES

- Watch these videos:
  - How does Netflix work? (https://www.youtube.com/watch?v=YXQpgAAeLM4)
  - How does Netflix work – An introduction to Netflix Stack (https://www.youtube.com/watch?v=8iWNUVe346o)
  - Building scalable web applications with Amazon Web Services (https://www.youtube.com/watch?v=DQkFgc2_M8o)

REFERENCES

- Software Architecture in Practice; Len Bass, Paul Clements, Rick Kazman