

Software Design and Architecture

[Quality Attributes] – Chapter 03, L02

Lecture Outlines

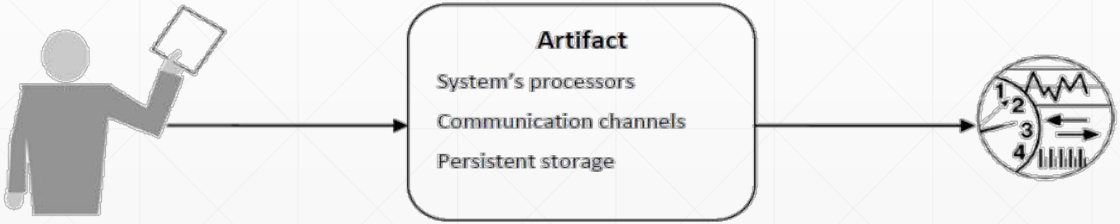
- Major Quality Attributes
  - Definition
  - Example with scenario
  - Tactics

Quality	Description
Usability	The degree of complexity involved when learning or using the system.
Modifiability	The degree of complexity involved when changing the system to fit current or future needs.
Security	The system's ability to protect and defend its information or information system.
Performance	The system's capacity to accomplish useful work under time and resource constraints.
Reliability	The system's failure rate.
Portability	The degree of complexity involved when adapting the system to other software or hardware environments.
Testability	The degree of complexity involved when verifying and validating the system's required functions.
Availability	The system's uptime.
Interoperability	The system's ability to collaborate with other software or hardware systems.

Major Quality Attributes

- **Availability:** is concerned with system failure and duration of system failures. (System failure means, when the system does not provide the service for which it was intended)
- **Modifiability:** is about the cost of change, both in time and money.
- **Performance:** is about timeliness. Events occur and the system must respond in a timely fashion.
- **Security:** is the ability of the system to prevent or resist unauthorized access while providing access to legitimate users. An attack is an attempt to breach security.
- **Testability:** refers to the ease with which the software can be made to demonstrate its faults. To be testable the system must control inputs and be able to observe outputs.
- **Usability:** is how easy it is for the user to accomplish tasks and what support the system provides for the user to accomplish this.

General availability scenario



Source	Stimulus	Environment	Response	Measure
Internal to system	Crash	Normal operation	Prevent the failure	Time interval available
External to system	Omission	Startup	Log the failure	Availability %
	Timing	Shutdown	Notify users / operators	Detection time
	No response	Repair mode	Disable source of failure	Repair time
	Incorrect response	Degraded (failsafe) mode	Temporarily unavailable	Degraded mode time interval
		Overloaded operation		

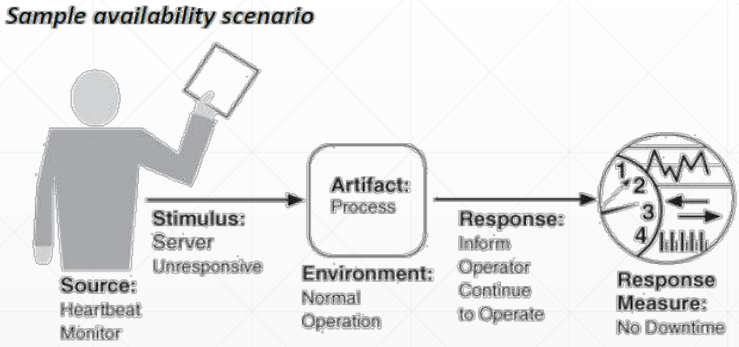
Availability – Scenario Example

Extra:

Response:

- ***Example on the response***: If a server becomes unresponsive, the response might involve:
  - Switching to a backup server.
  - Reallocating resources.
  - Or other corrective actions to ensure continued service.

- **Response Measure: It is the criteria used to evaluate the effectiveness of the response**. Also, we need it to determine how well the response addressed the problem, **focusing on the outcome and impact on the system's overall operation.**
- ***Example on the response measure***: For the server unresponsiveness issue, a response measure could be:
  - The time taken to restore service.
  - Or the system's ability to continue operating without downtime.



Extra: **6 Steps for Using Quality Attribute Scenarios**

1. **Source** → Who triggers the action (e.g., user, attacker, system).
2. **Stimulus** → The action or event that happens (e.g., request, failure, attack).
3. **Environment** → The situation in which it happens (e.g., under load, during maintenance).
4. **Artifact** → The part of the system affected (e.g., database, server, UI).
5. **Response** → How the system reacts as solution (e.g., log error, retry, deny access).
6. **Response Measure** → How success is judged (e.g., response time < 2 sec, 99% uptime).

Extra:

1.Fault Detection:

1. **Tactics**: Ping/Echo, Heartbeat, Exception
2. **Purpose**: These tactics are designed to identify faults promptly. For example, the Heartbeat monitors the pulse of the system to detect any abnormal behavior or failures.

2.Recovery-Preparation and Repair:

1. **Tactics**: Voting, Active Redundancy, Passive Redundancy, Spare
2. **Purpose**: These strategies prepare the system for a quick recovery.
   Active redundancy involves having a duplicate system running in parallel, ready to take over immediately,
   while passive redundancy kicks in only when the primary system fails.

3.Recovery Reintroduction:

1. **Tactics**: Shadow State, Resynchronization, Rollback
2. **Purpose**: Once the immediate fault is managed, these tactics help in bringing the system back to its normal state. As **Shadow** means the system keeps a *parallel copy* (a "shadow") of its current state while it is making changes. **Example:** A database keeps a shadow page (copy of the data page) when updating records. If the update fails, the system can roll back to the shadow copy without losing integrity. And **Resynchronization** means ensures that the restarted part has the latest valid data and is aligned with other running components. **Example:** In a distributed system, if one server crashes and comes back online, it resynchronizes its state (e.g., user sessions, cache, logs) with the other servers to ensure everyone has the same data. And **Rollback,** for instance, undoes any errors by restoring previous data states.

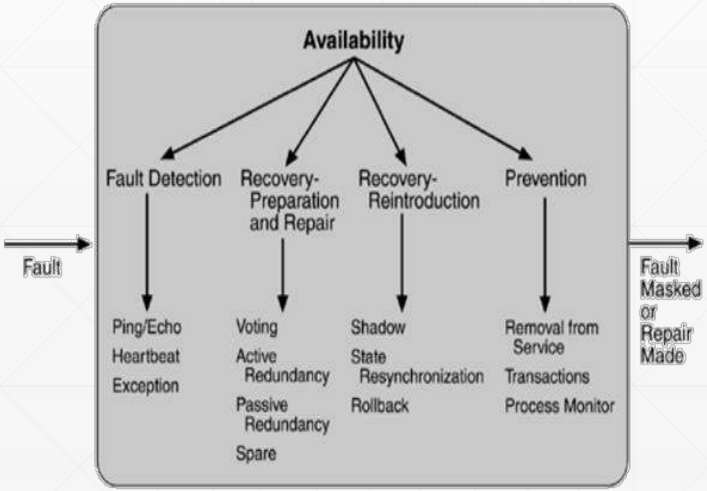
4.Prevention:

1. **Tactics**: Removal from Service, Transactions, Process Monitor
2. **Purpose**: These are long-term strategies aimed at preventing faults from reoccurring or minimizing their impact. Or Removing faulty components from service and monitoring ongoing processes help maintain system health and integrity. **Example**: If a payment gateway keeps failing, it is turned off and customers are sent to another working gateway.



Tactics are intended to control responses to stimuli.

availability tactics



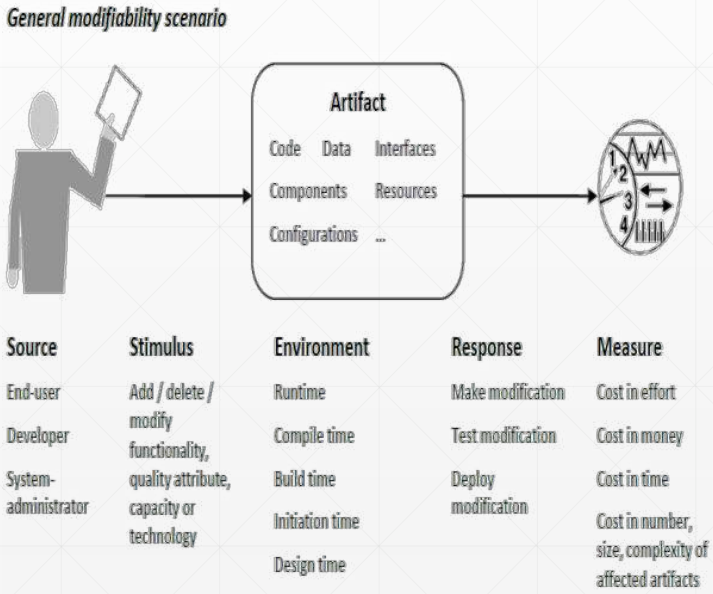
•Extra:

Modifiability – Scenario Example

1. Source: End User
2. Stimulus: Requests new functionality or changes in capacity/quality (e.g., faster login, dark mode, more users supported). cannot directly modify code or capacity. Instead, the end user requests new features, enhancements, or more capacity. Example: *"I want a dark mode"* or *"The system is too slow when 500 users log in."*
3. Environment: Runtime (system in use).
4. Artifact: Interfaces, resources, configurations.
5. Response: Developer/system admin implements changes, redeploys system.
6. Measure: Cost in money, time, downtime, user satisfaction.

1. Source: Developer
2. Stimulus: Adds/modifies/deletes functionality or quality attributes (e.g., refactoring code, improving security, optimizing performance).
3. Environment: Design time, compile time, build time.
4. Artifact: Code, components, data.
5. Response: Implement change → test → integrate.
6. Measure: Effort in coding/testing, complexity of changed artifacts.

1. Source :System Administrator
2. Stimulus: Modifies capacity or technology (e.g., scaling servers, upgrading database, adjusting configuration).
3. Environment: Initialization time or runtime.
4. Artifact: Configurations, resources, deployment environments.
5. Response: Apply configuration/upgrade → redeploy system → test stability.
6. Measure: Cost in downtime, effort, risk of errors, affected system resources.



Extra Scenarios:

1. Source: End user

•**Stimulus**: An end user requests a new feature or reports a bug.

•**Artifact**: The issue mainly affects user interfaces or functionalities.

•**Environment**: Feedback is considered during updates at runtime.

•**Response**: The software team updates the UI or functionalities based on the feedback.

•**Measure**: Success is measure by user satisfaction and feedback, decreased complaints

2. Source: Developer

•**Stimulus**: A developer wants to enhance the user interface (UI).

•**Artifact**: This change affects the code that controls the UI.

•**Environment**: The adjustments are made during the design phase.

•**Response**: The developer updates and tests the new UI changes.

•**Measure**: Success is evaluated based on the functionality and speed of implementation, aiming for completion and testing within three hours.

3. Source: system administrator

•**Stimulus**: The system faces increased load or requires enhanced security measures.

•**Artifact**: Focuses on scripts, network setups, and security settings

•**Environment**: Updates are made during live operation or planned downtime to avoid disruption.

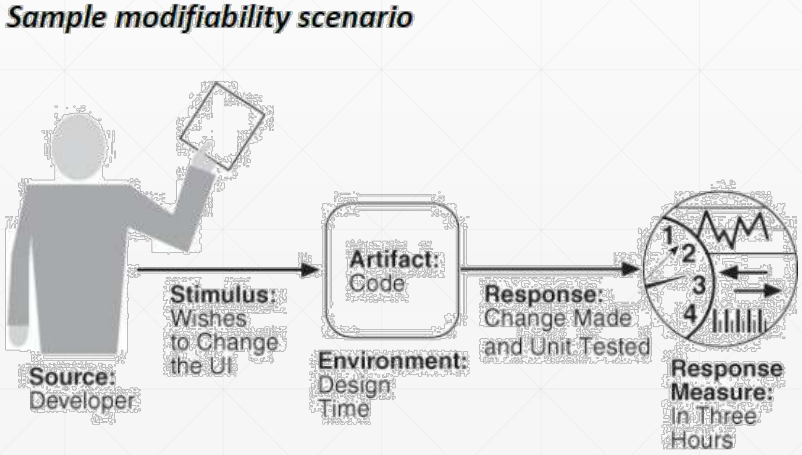
•**Response**: Implements hardware upgrades, adjusts network settings,

•**Measure**: Evaluates success through better system reliability, quicker system responses, and improved security outcomes.

Modifiability – Scenario Example

More information about general and sample Modifiability scenarios can be found in:

http://www.ece.ubc.ca/~matei/EECE417/BASS/ch04lev1sec4.html



Modifiability – tactics

•Extra:

- **Localize Changes**: When developers wish to change the UI, employing a strategy like "Generalize Module" or "Semantic Coherence" ensures that the change does not impact other parts of the software, focusing the change only on relevant components.

- **Prevention of Ripple Effect**: Example: You are updating the UI of a banking app to make it more user-friendly. any changes to the UI, like adding new visual elements or changing layouts, do not directly impact how account data is processed or retrieved.

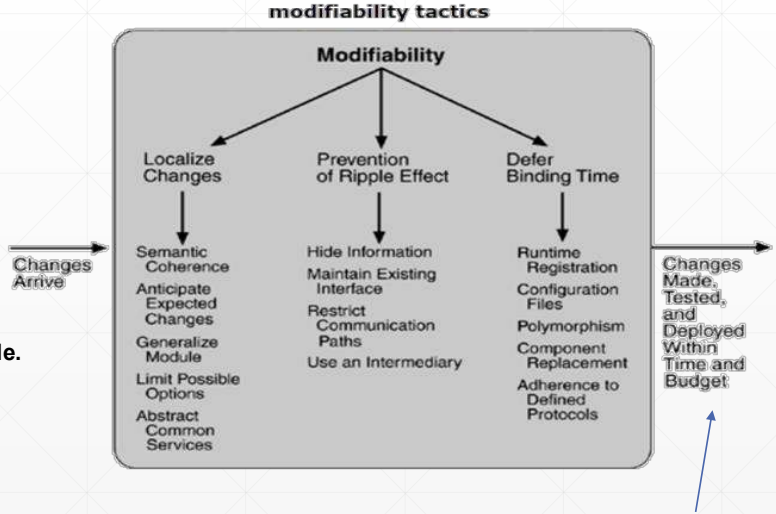
- **Defer Binding Time** = **delaying a decision as late as possible. Usually at runtime, not at design or compile time**.

Why we do it
- Easier to change
- No need to recompile or rebuild
- More flexibility

examples
- **UI Theme**
  - ❌ Hard binding: theme written in code
  - ✅ Deferred binding: theme read from a config file at runtime
- **Language setting**
  - ❌ Chosen at compile time
  - ✅ User selects language while app is running

summary

Deferred binding time means postponing decisions to runtime so the system stays flexible and easy to modify.



•**Response Measures in Context**: Measures of cost in including time, effort, and complexity of implementing changes are important for understanding efficiency.

Performance – Scenario Example

Extra:

1. **Source: it is the entity that generates the need for a response or action by the system.**

- Internal to the System: Such as a process within the system that requests resources.
- External to the System: Such as a user interaction or an external system's demand.

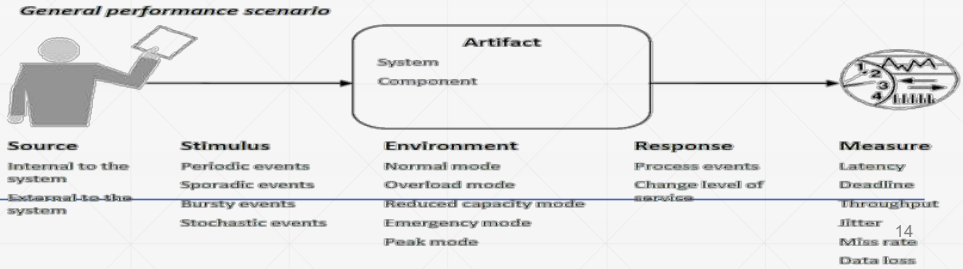
2. **Stimulus**:
- Periodic Events: Regularly occurring events, e.g., daily data backups.
- Sporadic Events: Occasional, unpredictable events, e.g., a user starting a large data upload unexpectedly.
- Bursty Events: Sudden changes in activity, e.g., many users logging in simultaneously.
- Stochastic Events: Random events that have probabilistic distributions, e.g., incoming network traffic.

3. **Artifact**:
- System Component: Any part of the system where the performance is measured or impacted, such as a server, database, or network component.

4. **Environment**:
- - Normal Mode: Standard operating conditions.
- Overload Mode: When the system is operating beyond its usual capacity.
- Reduced Capacity Mode: When the system is operating under resource constraints, possibly due to maintenance or partial failure.
- Emergency Mode: High-priority operations taking precedence during a crisis.
- Peak Mode: Highest expected operational demand.

5. **Response**:
- Process Events: Handling the incoming tasks or requests.
- Change Level of Service: Adjusting the performance levels, possibly by degrading non-essential services to maintain core functionality.

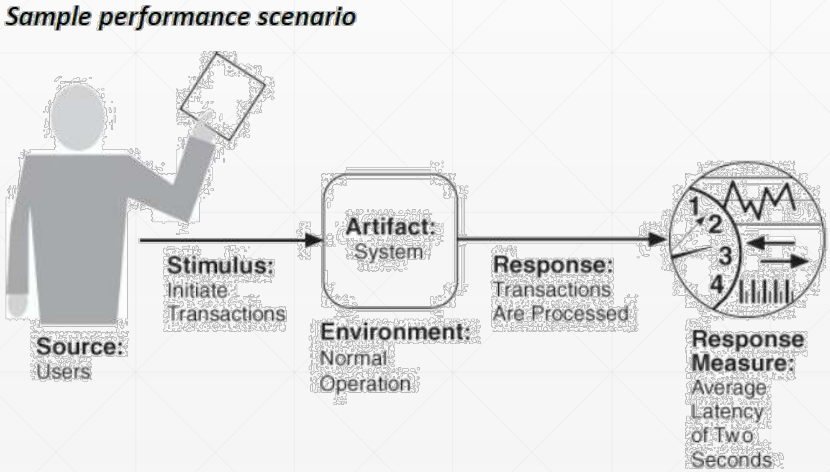
6. **Measure**:
- Latency: Delay before data starts moving.
  - Example: Waiting 2 seconds before a video starts playing.
- Deadline: Latest time a task must finish.
  - Example: A payment transaction must confirm within 5 seconds.
- Throughput: How much work/data is done per second.
  - Example: A server handles 1,000 requests per minute.
- Jitter: Inconsistency in response times.
  - Example: Sometimes a message arrives in 1s, other times in 4s.
- Miss Rate: How often deadlines or tasks are missed.
  - Example: out of 100 frames, 2 frames did not meet the speed requirement → **Miss Rate = 2%**.
- Data Loss: Data that never arrives.
  - Example: 5 out of 100 packets lost in a video call.



Performance – Scenario Example

More information about general and sample Performance scenarios can be found in:

http://www.ece.ubc.ca/~matei/EECE417/BASS/ch04lev1sec4.html



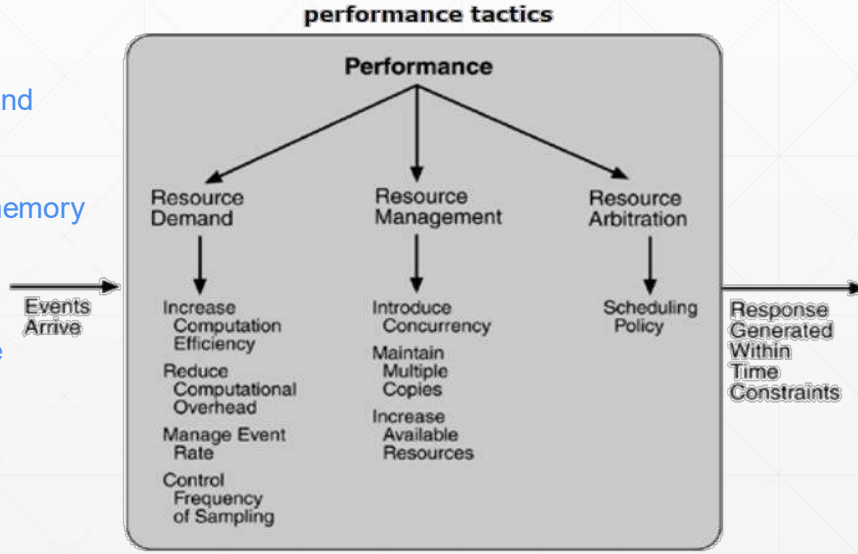
Performance – tactics

•**Extra:**

•**Resource Demand**: This can mean making the software run faster using less processing power and managing how often it checks or updates data.

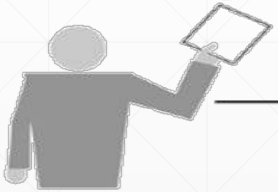
•**Resource Management**: Using resources like memory and processing power more effectively.

•**Resource Arbitration**: Deciding how to prioritize different tasks that need resources



Security – Scenario Example

General security scenario



Artifact

System services

Data within the system

Component / resource of the system

Data produced / consumed by the system



Source	Stimulus	Environment	Response	Measure
Identified user	Attempt to display data	Normal mode	Process events	Latency
Unknown user	Attempt to modify data	Overload mode	Change level of service	Deadline
Hacker from outside the organization	Attempt to delete data	Reduced capacity mode		Throughput
Hacker from inside the organization	Access system services	Emergency mode		Jitter
		Peak mode		Miss rate
				Data loss

Security – Scenario Example

Extra:

Source: A (Dissatisfied person) attempting to modify payroll data from a remote location.

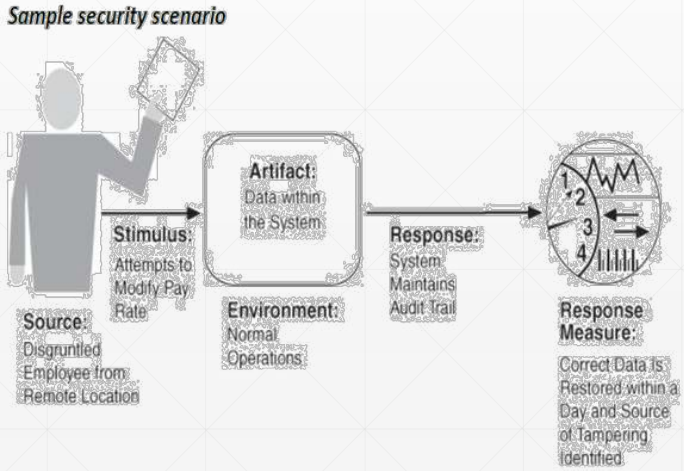
Stimulus: The attempt to modify the pay rate data.

Artifact: The specific data targeted by the attack within the system.

Environment: The system operates under normal conditions when the attack occurs.

Response: The system's response includes maintaining an audit trail, which helps in tracking and recording the sequence of activities.

Response Measure: The effectiveness of the response is measured by how quickly the correct data is restored and the identification of the source of tampering.



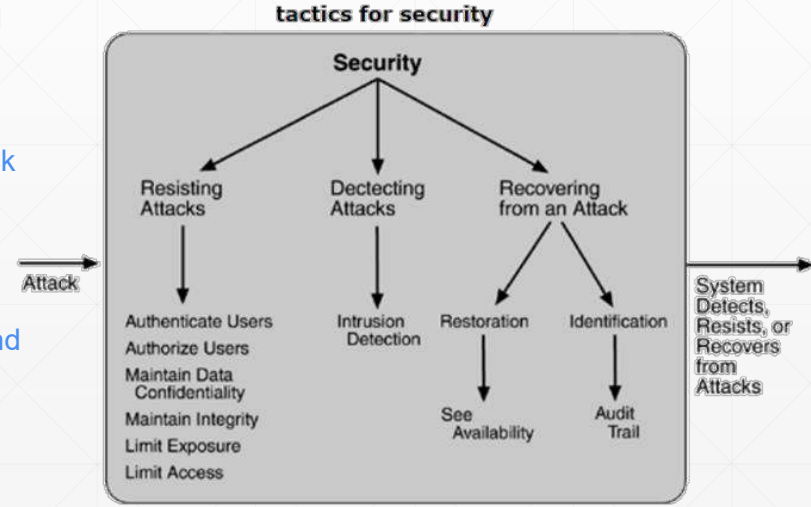
Extra

Security – tactics

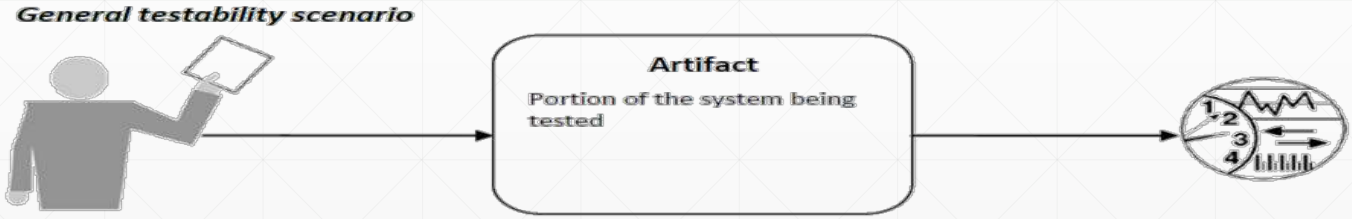
Resisting Attacks: Methods to prevent unauthorized access or modifications. This includes authenticating and authorizing users, maintaining data confidentiality and integrity, and limiting access and exposure.

Detecting Attacks: Techniques to identify when an attack is occurring, by intrusion detection systems.

Recovering from an Attack: Steps to restore system functionality and data integrity after an attack. This involves restoration, identification of the threat source, and maintaining an audit trail.



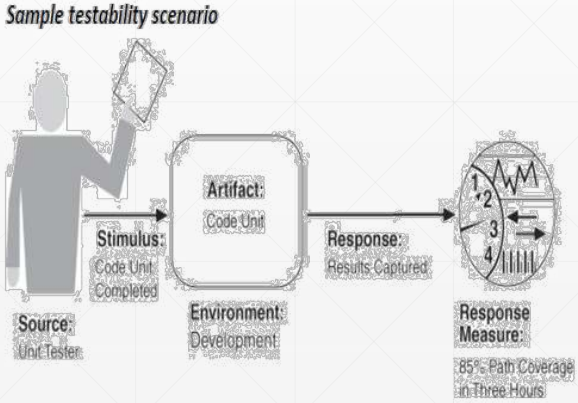
Testability – Scenario Example



Source	Stimulus	Environment	Response	Measure
Unit tester	Execution of tests due to completion of code increment	Design time	Execute test suite & capture results	Effort to find fault
Integration tester		Development time	Capture cause of fault	Effort to achieve coverage %
System tester		Compile time	Control & monitor state of the system	Probability of fault being revealed by next test
Acceptance tester		Integration time		Time to perform tests
End user		Deployment time		Effort to detect faults
Automated testing tools		Run time		

Testability – Scenario Example

•**Extra:**
•**Source**: The Unit Tester is the individual responsible for testing.
•**Stimulus**: The completion of a code unit, prompting the test.
•**Artifact**: The specific code unit that has been completed and is being tested.
•**Environment**: Development environment where testing typically occurs.
•**Response**: Results of the testing are captured.
•**Response Measure**: Evaluates the effectiveness of the testing with metrics such as 85% path coverage within three hours.

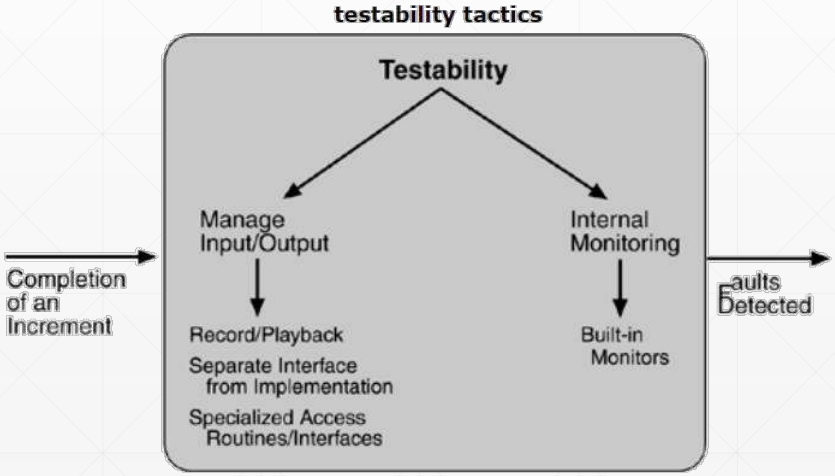


Testability – tactics

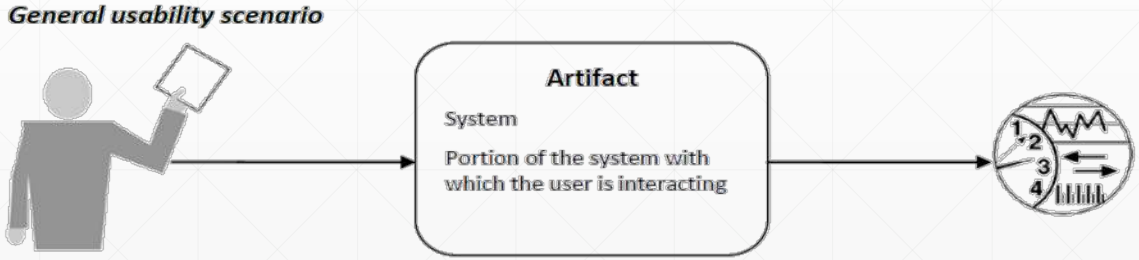
Extra

This diagram outlines general strategies that enhance the testability of a system:

- **Manage Input/Output**: Includes techniques like record/playback and separating interface from implementation, which simplify testing by allowing testers to manage how data enters and exits the system.
- **Internal Monitoring**: Involves built-in monitors that help in observing the system's internal state, providing valuable data during testing.



Usability – Scenario Example



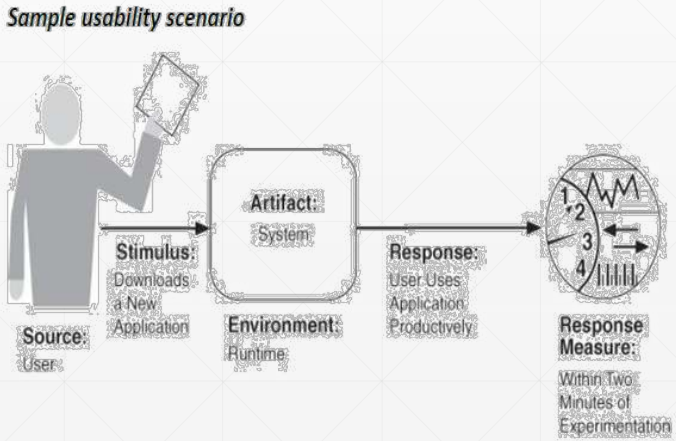
Source	Stimulus	Environment	Response	Measure
End user (possibly special role)	Use the system efficiently	Runtime	Provide features needed	Task time
	Learn to use the system	Configuration time	Anticipate the user's needs	Number of errors
	Minimize impact of errors			Number of tasks accomplished
	Adapt the system			User satisfaction
	Configure the system			Gain of user knowledge

Usability – Scenario Example

Extra:

This diagram represents a usability scenario where a user downloads and begins using a new application. The key elements are:

1. **Source**: Users who initiate the test by downloading the app.
2. **Stimulus**: The action of downloading and starting the app.
3. **Artifact**: The overall system that supports and runs the app.
4. **Environment**: Occurs in real-time during application use.
5. **Response**: The user is able to use the app effectively almost immediately.
6. **Response Measure**: Success is measured by the user achieving productivity within two minutes of starting the app, indicating high usability.



Usability – tactics

Extra:

1. Separate User Interface

Explanation: Keeping the user interface distinct from system operations.

Example: A blog platform lets you customize your blog's design through a visual editor without needing to touch the underlying code.

2. Support User Initiative

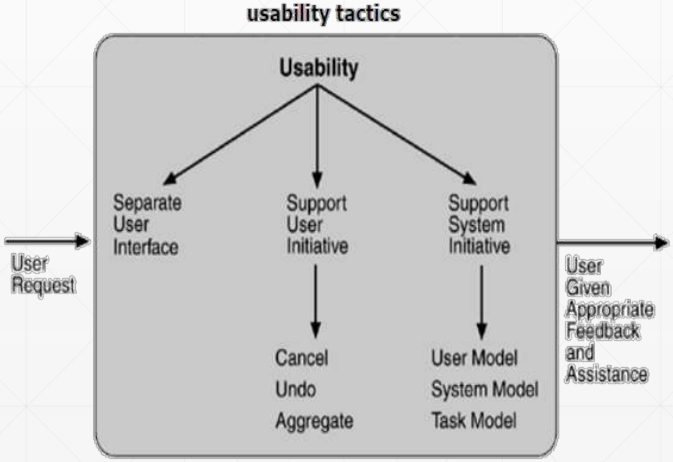
Explanation: Tools that help users manage their actions.

- **Cancel**: In an online shopping cart, users can remove items before checkout.
- **Undo**: In a document editor, users can revert changes by clicking the "undo" button.
- **Aggregate**: A fitness app compiles weekly activity into a summary report for easy review.

3. Support System Initiative

Explanation: The system anticipates and reacts to user needs based on user behavior and tasks.

- **User Model**: A streaming service recommends movies based on your viewing history.
- **System Model**: An email client automatically syncs your most important emails first.
- **Task Model**: A task management app sends reminders for deadlines based on project timelines.



Summary

- Major Quality Attributes
  - Definition
  - General and sample scenario examples
  - Tactics
- Next … Architecture Design Patterns

- https://www.youtube.com/watch?time_continue=85&v=rKwSIih_U74&embeds_referring_euri=https%3A%2F%2Fchatgpt.com%2F&source_ve_path=MjM4NTE

Extra: Question List one design tactic that you can use to achieve each of the following quality attributes:

Quality Attribute	Tactics
Security	Resisting Attacks, Detecting Attacks, Authenticate users
Testability	Separate interface from implementation
Modifiability	Localize changes, Encapsulation, Hide information
Performance	Concurrency, Increase available resources.