CptS 487
Software Design and Architecture

Lesson 18 (pt3)

Quality Attributes 3

Security & Testability

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Instructor: Bolong Zeng

Outline

- Security
- Testability

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."
- The CIA characteristics:
  - Confidentiality
  - Integrity
  - Availability
- Supportive characteristics:
  - Authentication
  - Nonrepudiation
  - Authorization

Discussion of Security

- General Scenario

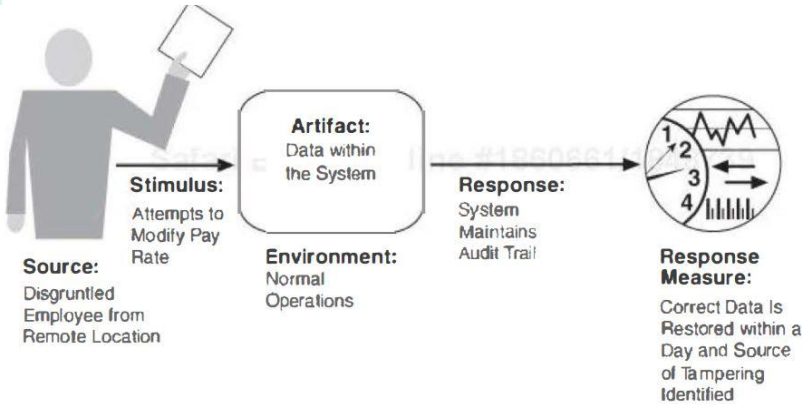

Figure 9.1. Sample concrete security scenario

Table 9.1. Security General Scenario

Portion of Scenario	Possible Values
Source	Human or another system which may have been previously identified (either correctly or incorrectly) or may be currently unknown. A human attacker may be from outside the organization or from inside the organization.
Stimulus	Unauthorized attempt is made to display data, change or delete data, access system services, change the system's behavior, or reduce availability.
Artifact	System services, data within the system, a component or resources of the system, data produced or consumed by the system
Environment	The system is either online or offline; either connected to or disconnected from a network; either behind a firewall or open to a network; fully operational, partially operational, or not operational.
Response	Transactions are carried out in a fashion such that ▪ Data or services are protected from unauthorized access. ▪ Data or services are not being manipulated without authorization. ▪ Parties to a transaction are identified with assurance. ▪ The parties to the transaction cannot repudiate their involvements. ▪ The data, resources, and system services will be available for legitimate use. The system tracks activities within it by ▪ Recording access or modification ▪ Recording attempts to access data, resources, or services ▪ Notifying appropriate entities (people or systems) when an apparent attack is occurring
Response Measure	One or more of the following: ▪ How much of a system is compromised when a particular component or data value is compromised ▪ How much time passed before an attack was detected ▪ How many attacks were resisted ▪ How long does it take to recover from a successful attack ▪ How much data is vulnerable to a particular attack

- Concrete Scenario

Discussion of Security

- Tactics
  - 4 categories
    - Detect
    - Resist
    - React
    - Recover

Figure 9.2. The goal of security tactics

Detect Attacks

- Detect intrusion
- Detect service
- Verify message integrity
- Detect message delay

Resist Attacks

- Identify actors
- Authenticate actors
- Authorize actors
- Limit access
- Limit exposure
- Encrypt data
- Separate entities
- Change default settings

React to Attacks

- Revoke access
- Lock computer
- Inform actors

Recover from Attacks

- Maintain audit trail
- Restore tactics for Availability

Summary on Tactics

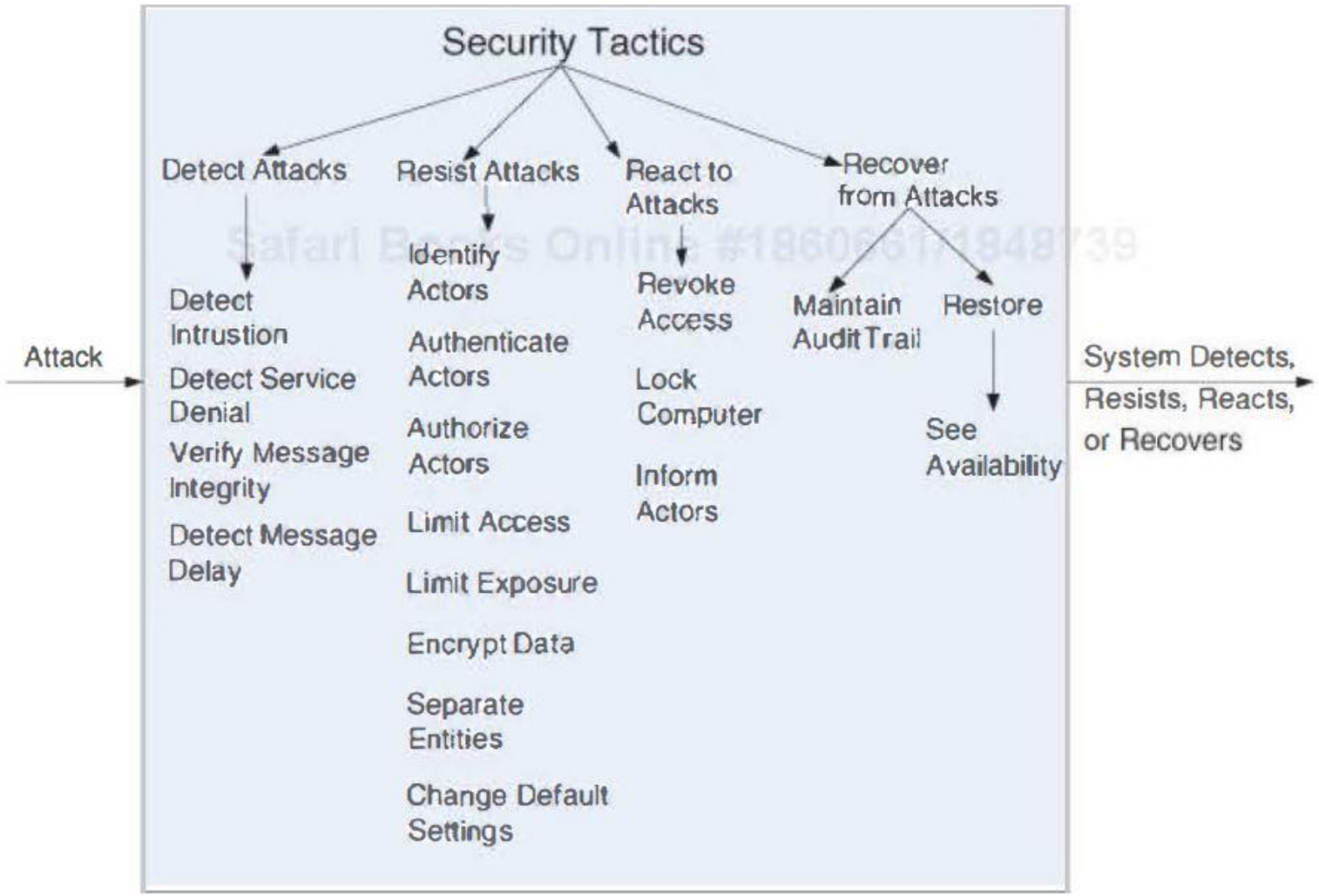

Figure 9.3. Security tactics

Design Checklist for Security

Table 9.2. Checklist to Support the Design and Analysis Process for Security

Category	Checklist
Allocation of Responsibilities	Determine which system responsibilities need to be secure. For each of these responsibilities, ensure that additional responsibilities have been allocated to do the following: ▪ Identify the actor ▪ Authenticate the actor ▪ Authorize actors ▪ Grant or deny access to data or services ▪ Record attempts to access or modify data or services ▪ Encrypt data ▪ Recognize reduced availability for resources or services and inform appropriate personnel and restrict access ▪ Recover from an attack ▪ Verify checksums and hash values
Coordination Model	Determine mechanisms required to communicate and coordinate with other systems or individuals. For these communications, ensure that mechanisms for authenticating and authorizing the actor or system, and encrypting data for transmission across the connection, are in place. Ensure also that mechanisms exist for monitoring and recognizing unexpectedly high demands for resources or services as well as mechanisms for restricting or terminating the connection.
Data Model	Determine the sensitivity of different data fields. For each data abstraction: ▪ Ensure that data of different sensitivity is separated. ▪ Ensure that data of different sensitivity has different access rights and that access rights are checked prior to access. ▪ Ensure that access to sensitive data is logged and that the log file is suitably protected. ▪ Ensure that data is suitably encrypted and that keys are separated from the encrypted data. ▪ Ensure that data can be restored if it is inappropriately modified.

Design Checklist for Security (Cont.)

Mapping among Architectural Elements	Determine how alternative mappings of architectural elements that are under consideration may change how an individual or system may read, write, or modify data; access system services or resources; or reduce availability to system services or resources. Determine how alternative mappings may affect the recording of access to data, services or resources and the recognition of unexpectedly high demands for resources. For each such mapping, ensure that there are responsibilities to do the following: • Identify an actor • Authenticate an actor • Authorize actors • Grant or deny access to data or services • Record attempts to access or modify data or services • Encrypt data • Recognize reduced availability for resources or services, inform appropriate personnel, and restrict access • Recover from an attack
Resource Management	Determine the system resources required to identify and monitor a system or an individual who is internal or external, authorized or not authorized, with access to specific resources or all resources. Determine the resources required to authenticate the actor, grant or deny access to data or resources, notify appropriate entities (people or systems), record attempts to access data or resources, encrypt data, recognize inexplicably high demand for resources, inform users or systems, and restrict access. For these resources consider whether an external entity can access a critical resource or exhaust a critical resource; how to monitor the resource; how to manage resource utilization; how to log resource utilization; and ensure that there are sufficient resources to perform the necessary security operations. Ensure that a contaminated element can be prevented from contaminating other elements. Ensure that shared resources are not used for passing sensitive data from an actor with access rights to that data to an actor without access rights to that data.
Binding Time	Determine cases where an instance of a late-bound component may be untrusted. For such cases ensure that late-bound components can be qualified; that is, if ownership certificates for late-bound components are required, there are appropriate mechanisms to manage and validate them; that access to late-bound data and services can be managed; that access by late-bound components to data and services can be blocked; that mechanisms to record the access, modification, and attempts to access data or services by late-bound components are in place; and that system data is encrypted where the keys are intentionally withheld for late-bound components
Choice of Technology	Determine what technologies are available to help user authentication, data access rights, resource protection, and data encryption. Ensure that your chosen technologies support the tactics relevant for your security needs.

Testability

- "Software testability refers to the ease with which software can be made to demonstrate its faults through testing."
- Model of testing
  - Oracle
    - Decides whether the output is correct
    - Output can be "produced value" or "internal state" of the program
      - Somewhat corresponding to black-box and white-box testing.

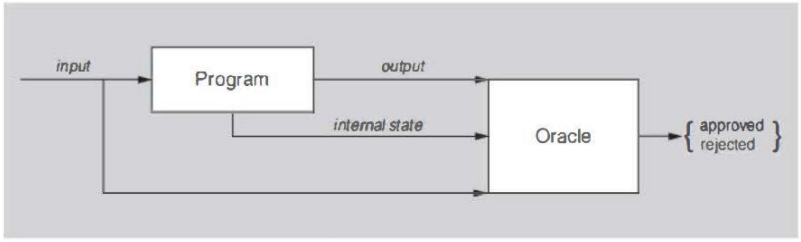

Figure 10.1. A model of testing

- Recommended reading:
  - [BASS] Chapter 10 sidebar: "Netflix's Simian Army"

Discussion of Testability

- General Scenario

Table 10.1. Testability General Scenario

Portion of Scenario	Possible Values
Source	Unit testers, integration testers, system testers, acceptance testers, end users, either running tests manually or using automated testing tools
Stimulus	A set of tests is executed due to the completion of a coding increment such as a class layer or service, the completed integration of a subsystem, the complete implementation of the whole system, or the delivery of the system to the customer.
Environment	Design time, development time, compile time, integration time, deployment time, run time
Artifacts	The portion of the system being tested
Response	One or more of the following: execute test suite and capture results, capture activity that resulted in the fault, control and monitor the state of the system
Response Measure	One or more of the following: effort to find a fault or class of faults, effort to achieve a given percentage of state space coverage, probability of fault being revealed by the next test, time to perform tests, effort to detect faults, length of longest dependency chain in test, length of time to prepare test environment, reduction in risk exposure (size(loss) × prob(loss))

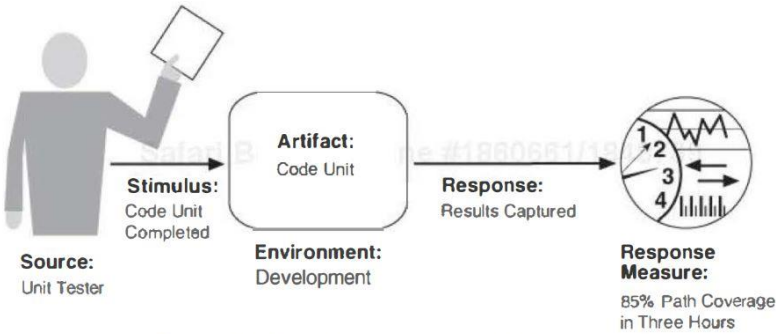

Figure 10.2. Sample concrete testability scenario

- Concrete Scenario

Discussion of Testability

- Tactics
  - Goal: allow for easier testing when an increment of software development is completed.
  - 2 categories
    - Control and observe system state
    - Limit complexity

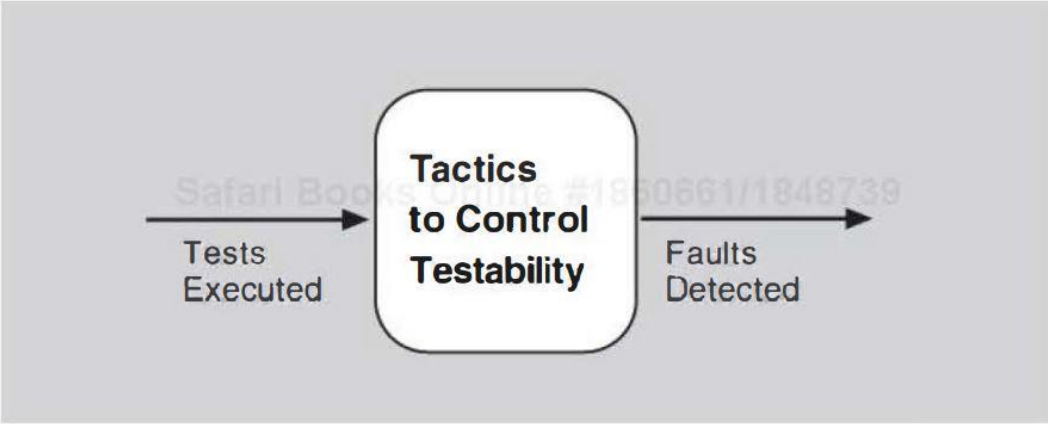

Figure 10.3. The goal of testability tactics

Control and Observe System State

- Specialized interfaces
- Record/playback
- Localize state storage
- Abstract data sources
- Sandbox
- Executable assertions

Limit Complexity

- Limit structural complexity
- Limit nondeterminism

Summary on Tactics

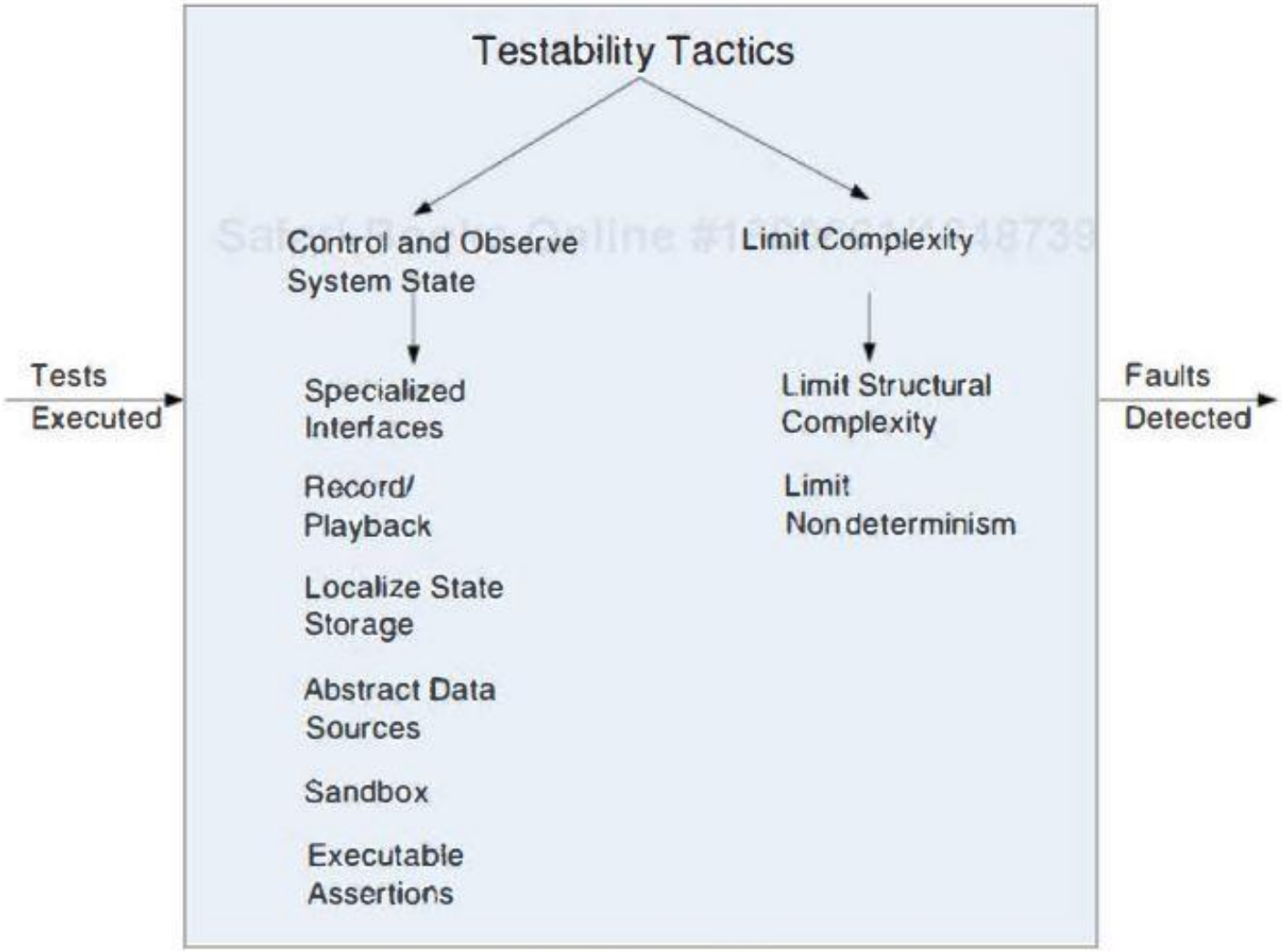

Figure 10.4. Testability tactics

Design Checklist for Testability

Table 10.2. Checklist to Support the Design and Analysis Process for Testability

Category	Checklist
Allocation of Responsibilities	Determine which system responsibilities are most critical and hence need to be most thoroughly tested. Ensure that additional system responsibilities have been allocated to do the following: ▪ Execute test suite and capture results (external test or self-test) ▪ Capture (log) the activity that resulted in a fault *or* that resulted in unexpected (perhaps emergent) behavior that was not necessarily a fault ▪ Control and observe relevant system state for testing Make sure the allocation of functionality provides high cohesion, low coupling, strong separation of concerns, and low structural complexity.
Coordination Model	Ensure the system's coordination and communication mechanisms: ▪ Support the execution of a test suite and capture the results within a system or between systems ▪ Support capturing activity that resulted in a fault within a system or between systems ▪ Support injection and monitoring of state into the communication channels for use in testing, within a system or between systems ▪ Do not introduce needless nondeterminism
Data Model	Determine the major data abstractions that must be tested to ensure the correct operation of the system. ▪ Ensure that it is possible to capture the values of instances of these data abstractions ▪ Ensure that the values of instances of these data abstractions can be set when state is injected into the system, so that system state leading to a fault may be re-created ▪ Ensure that the creation, initialization, persistence, manipulation, translation, and destruction of instances of these data abstractions can be exercised and captured
Mapping among Architectural Elements	Determine how to test the possible mappings of architectural elements (especially mappings of processes to processors, threads to processes, and modules to components) so that the desired test response is achieved and potential race conditions identified. In addition, determine whether it is possible to test for illegal mappings of architectural elements.
Resource Management	Ensure there are sufficient resources available to execute a test suite and capture the results. Ensure that your test environment is representative of (or better yet, identical to) the environment in which the system will run. Ensure that the system provides the means to do the following: ▪ Test resource limits ▪ Capture detailed resource usage for analysis in the event of a failure ▪ Inject new resource limits into the system for the purposes of testing ▪ Provide virtualized resources for testing
Binding Time	Ensure that components that are bound later than compile time can be tested in the late-bound context. Ensure that late bindings can be captured in the event of a failure, so that you can re-create the system's state leading to the failure. Ensure that the full range of binding possibilities can be tested.
Choice of Technology	Determine what technologies are available to help achieve the testability scenarios that apply to your architecture. Are technologies available to help with regression testing, fault injection, recording and playback, and so on? Determine how testable the technologies are that you have chosen (or are considering choosing in the future) and ensure that your chosen technologies support the level of testing appropriate for your system. For example, if your chosen technologies do not make it possible to inject state, it may be difficult to re-create fault scenarios.

Conclusion

- There are many more universal/specific QAs.
  - [BASS] Chapter 12 for references.

- Ultimate purpose of QAs
  - Provide guidelines for software engineering practice
  - Tactics to achieve the qualities help consolidate the architecture
  - Be able to analysis, measure and evaluate the qualities.

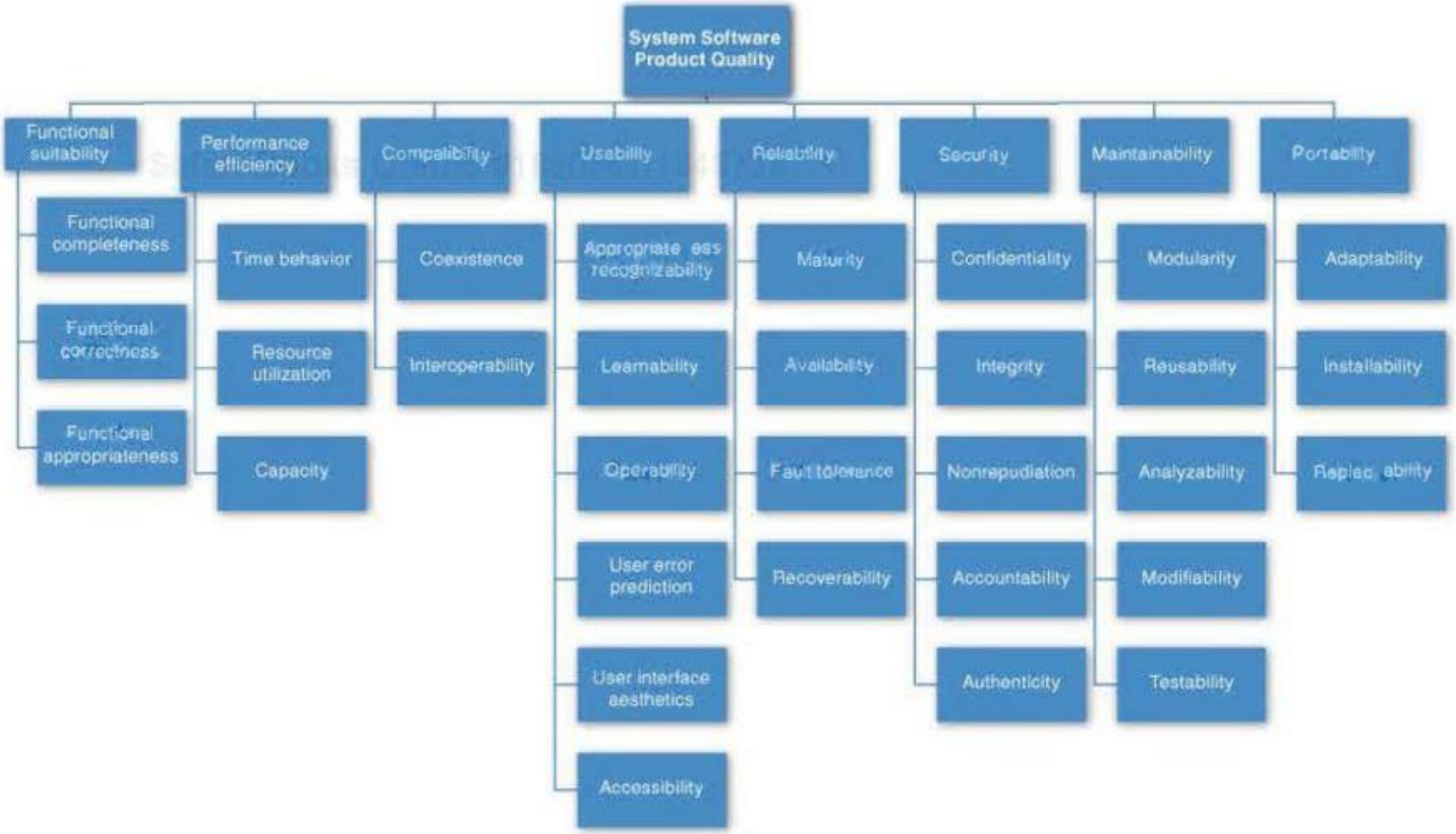

Figure 12.1. The ISO/IEC FCD 25010 product quality standard