AMRITA VISHWA VIDYAPEETHAM

19CSE314 Software Engineering

Testing strategies and tactics

Testing strategies and tactics

Pressman R S, Bruce R.Maxim, “Software engineering - A Practitioner’s Approach”, Eighth Edition, Tata McGraw-Hill, 2014
Chapter 22,23,25,26

Topics Covered

Topic	**Text Book Reference**
A STRATEGIC APPROACH TO SOFTWARE TESTING	**22.1, 22.2**
TEST STRATEGIES FOR CONVENTIONAL SOFTWARE	**22.3**
Test strategies for Web Apps	**22.5**
Test strategies for Mobile Apps	**22.6**
Validation Testing	**22.7**
System Testing	**22.8**

Software Testing Strategies

- A strategy for software testing provides a road map that describes the steps to be conducted as part of testing, when these steps are planned and then undertaken, and how much effort, time, and resources will be required
- Therefore, any testing strategy must incorporate test planning, test-case design, test execution, and resultant data collection and evaluation
- A software testing strategy should be flexible enough to promote a customized testing approach
- At the same time, it must be rigid enough to encourage reasonable planning and management tracking as the project progresses

A STRATEGIC APPROACH TO SOFTWARE TESTING

- A number of software testing strategies have been proposed in the literature.
- All provide you with a template for testing and all have the following generic characteristics:
  - To perform effective testing, you should conduct effective technical reviews. By doing this, many errors will be eliminated before testing commences
  - Testing begins at the component level and works “outward” toward the integration of the entire computer-based system
  - Different testing techniques are appropriate for different software engineering approaches and at different points in time.
  - Testing is conducted by the developer of the software and (for large projects) an independent test group
  - Testing and debugging are different activities, but debugging must be accommodated in any testing strategy

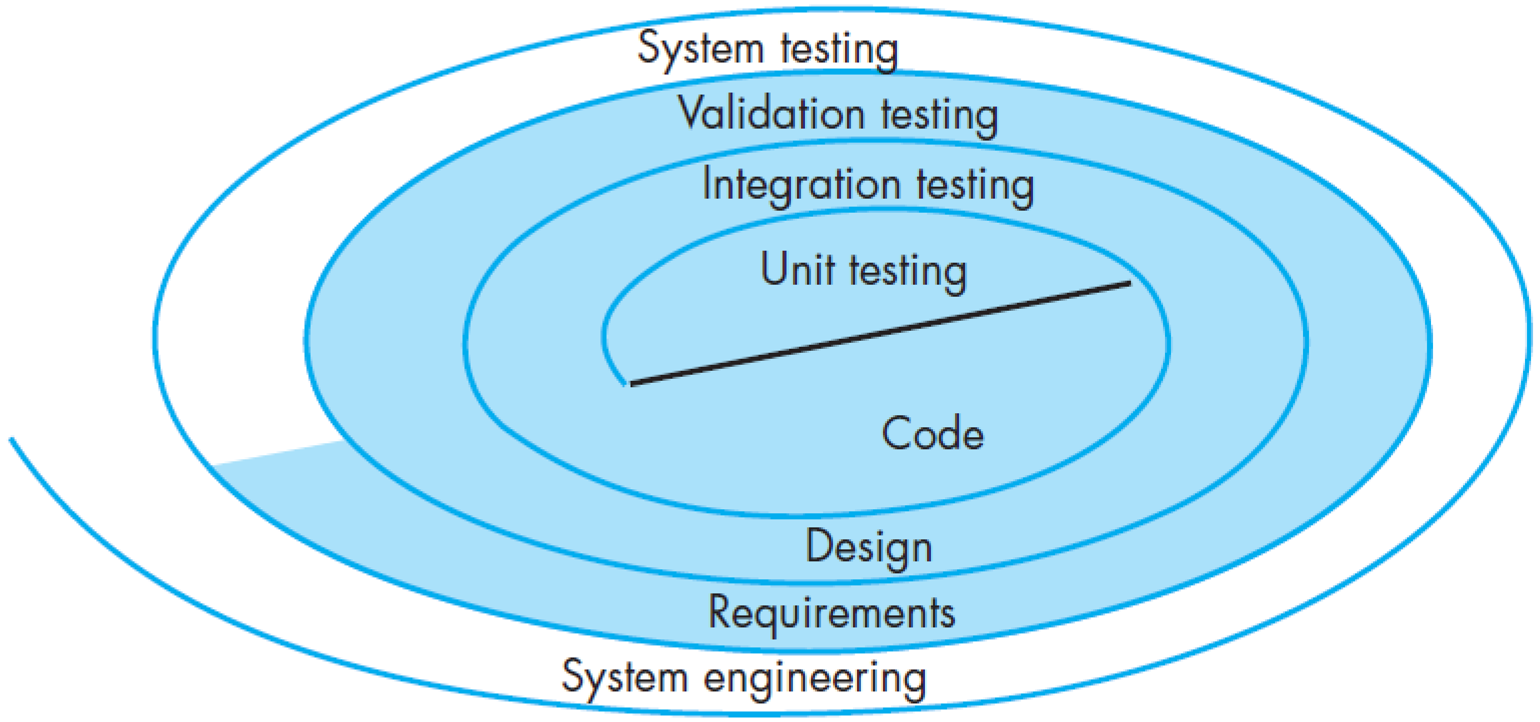
Verification and Validation (V&V)

- *Verification* refers to the set of tasks that ensure that software correctly implements a specific function
- *Validation* refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements
- Boehm [Boe81] states this another way:
  - Verification: “Are we building the product right?”
  - Validation: “Are we building the right product?”

Verification and Validation (V&V)

- Verification and validation includes a wide array of SQA activities:
  - technical reviews
  - quality and configuration audits
  - performance monitoring
  - Simulation
  - feasibility study
  - documentation review
  - database review
  - algorithm analysis
  - development testing
  - usability testing
  - qualification testing
  - acceptance testing
  - installation testing
- Although testing plays an extremely important role in V&V, many other activities are also necessary

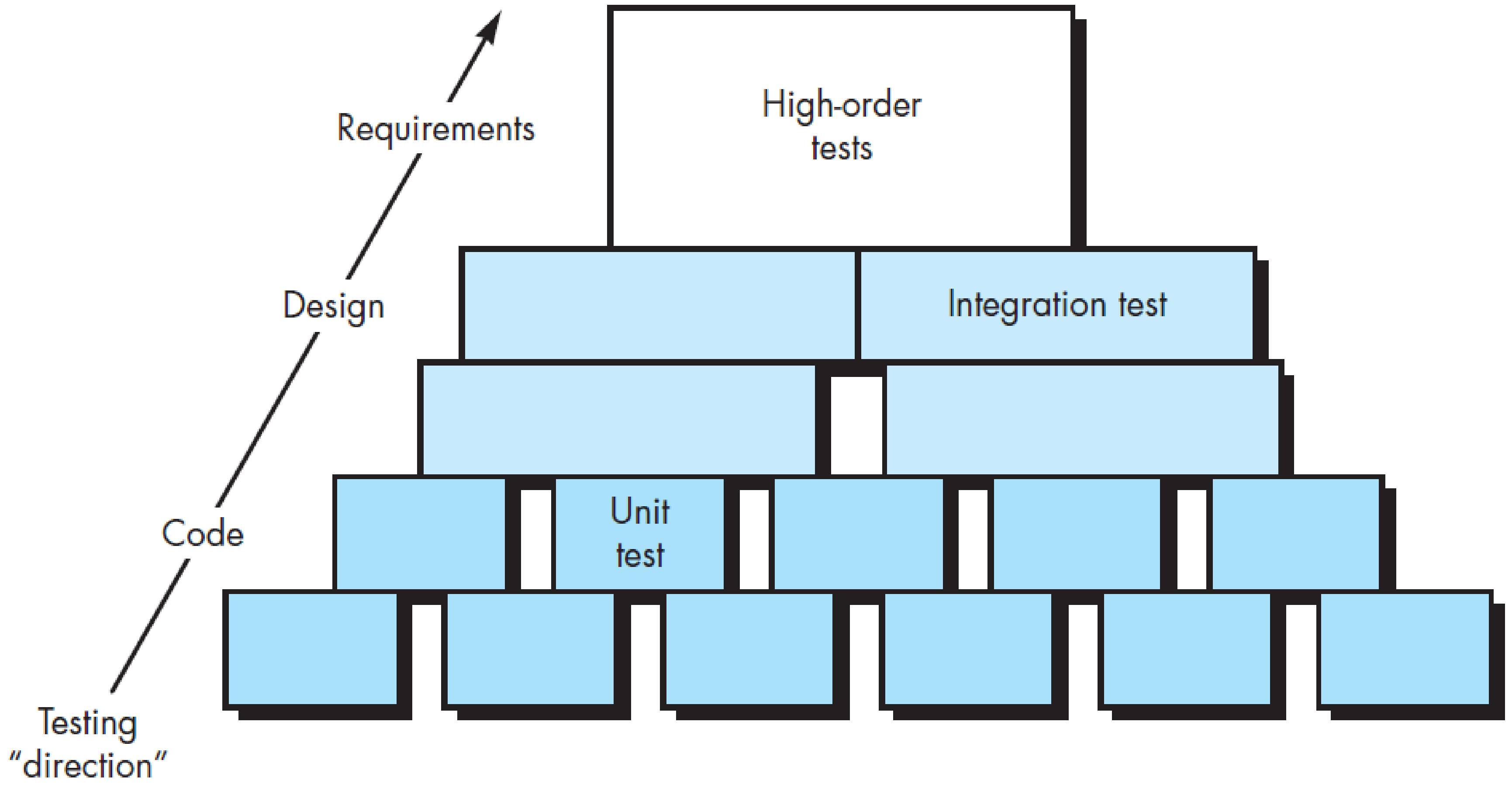
Software Testing Strategy



Discussion

- Is it always possible to develop a strategy for testing software that uses the sequence of testing steps described in last slide?
- What possible complications might arise for embedded systems?

Software testing steps



Strategic Issues

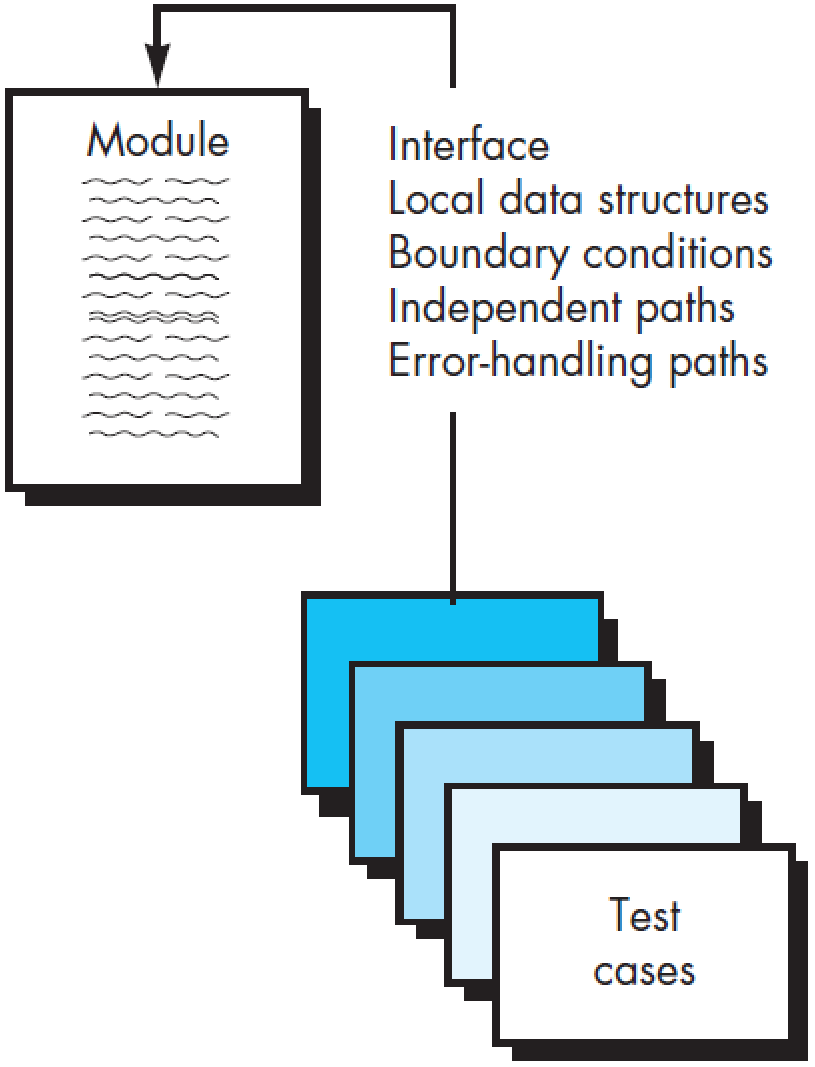
- specify product requirements in a quantifiable manner long before testing commences
- state testing objectives explicitly
- Understand the users of the software and develop a profile for each user category,
- Develop a testing plan that emphasizes “rapid cycle testing,”
- build “robust” software that is designed to test itself
- use effective technical reviews as a filter prior to testing
- Conduct technical reviews to assess the test strategy and test cases themselves, and
- develop a continuous improvement approach (Chapter 37) for the testing process

Test Strategies for conventional software

Unit Testing

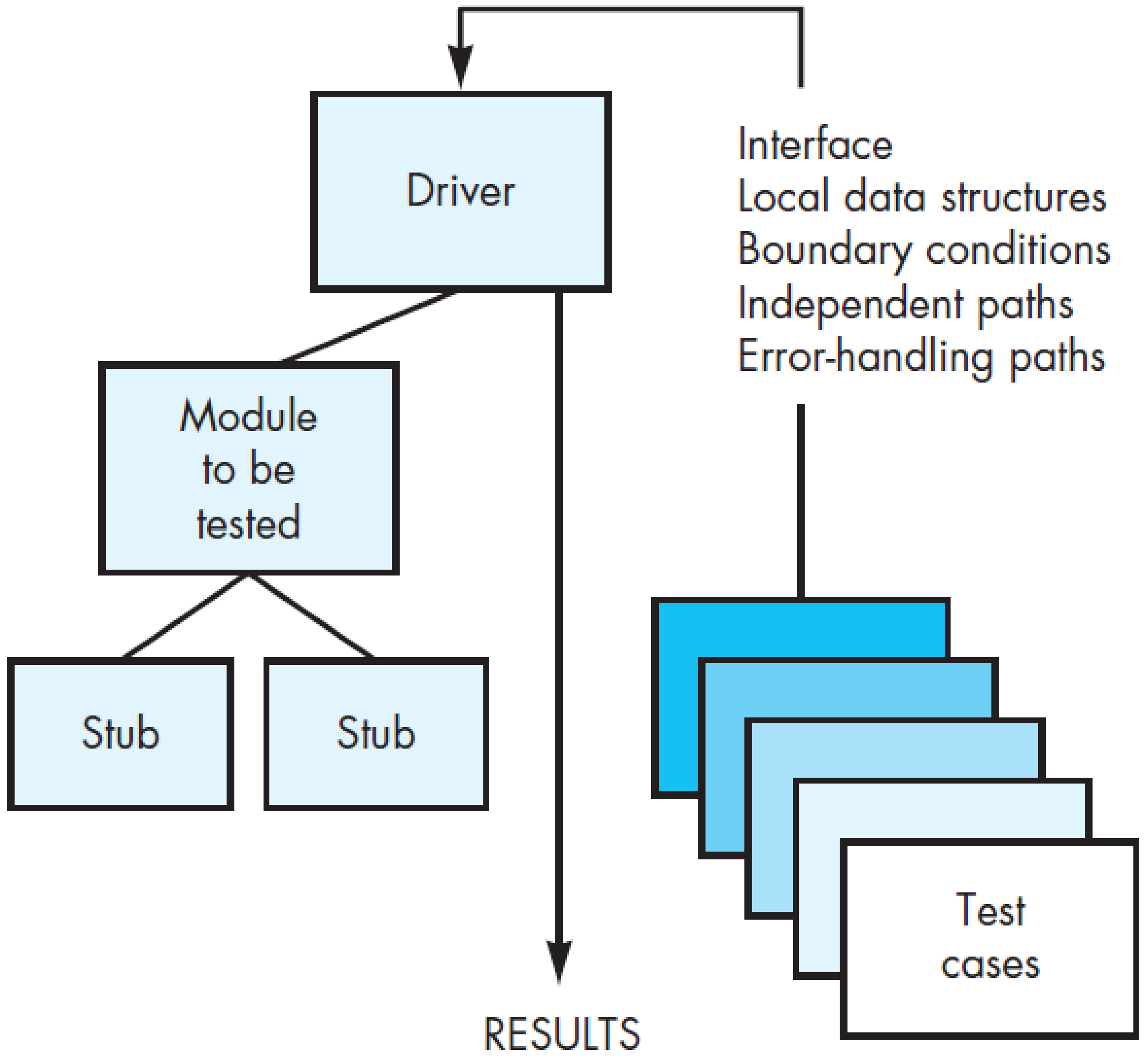
Integration Testing

Unit Testing



- *Unit testing* focuses verification effort on the smallest unit of software design— the software component or module
- Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module
- The relative complexity of tests and the errors those tests uncover is limited by the constrained scope established for unit testing
- The unit test focuses on the internal processing logic and data structures within the boundaries of a component
- This type of testing can be conducted in parallel for multiple components

Unit test environment



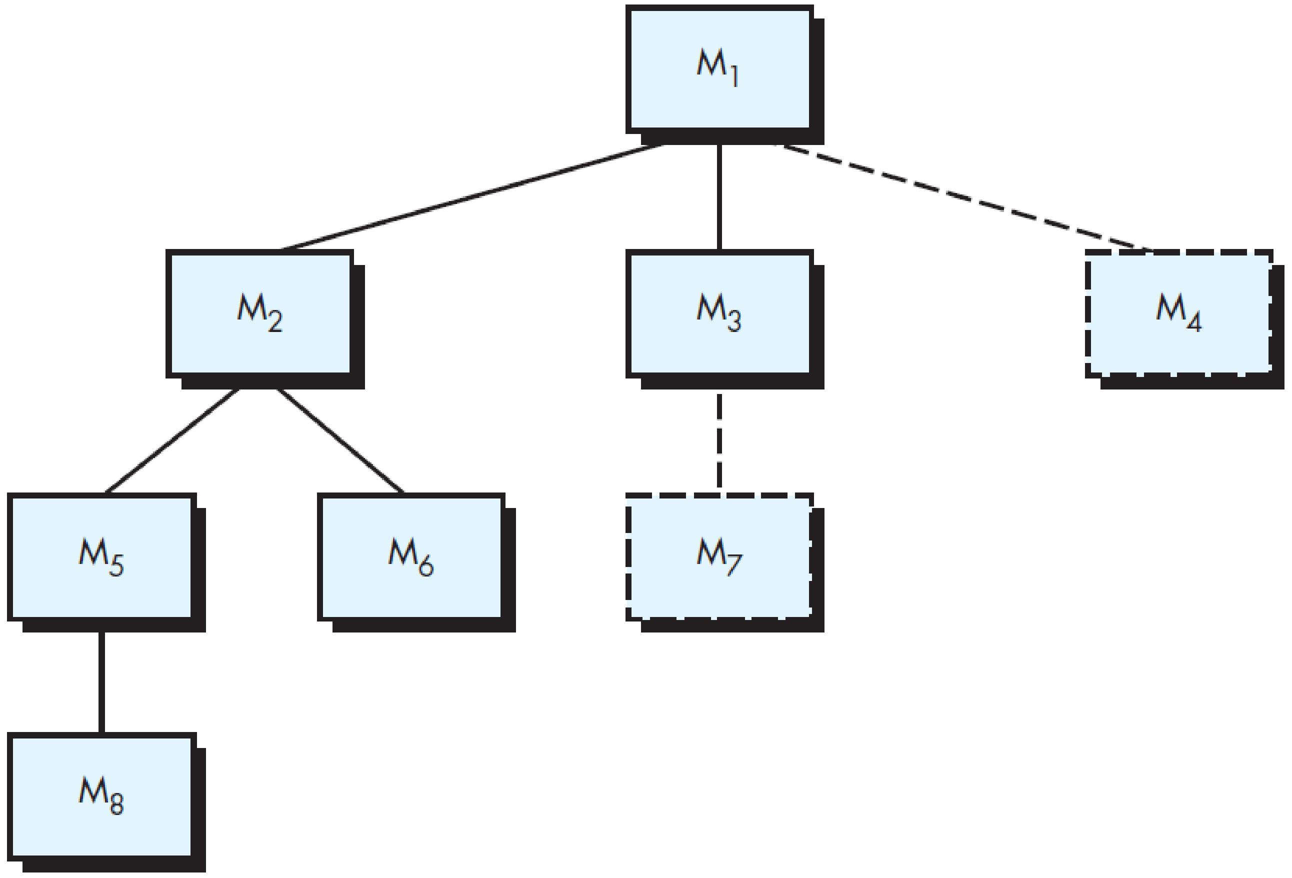
- In most applications a *driver* is nothing more than a “main program” that accepts test-case data, passes such data to the component (to be tested), and prints relevant results
- *Stubs* serve to replace modules that are subordinate (invoked by) the component to be tested

Integration Testing

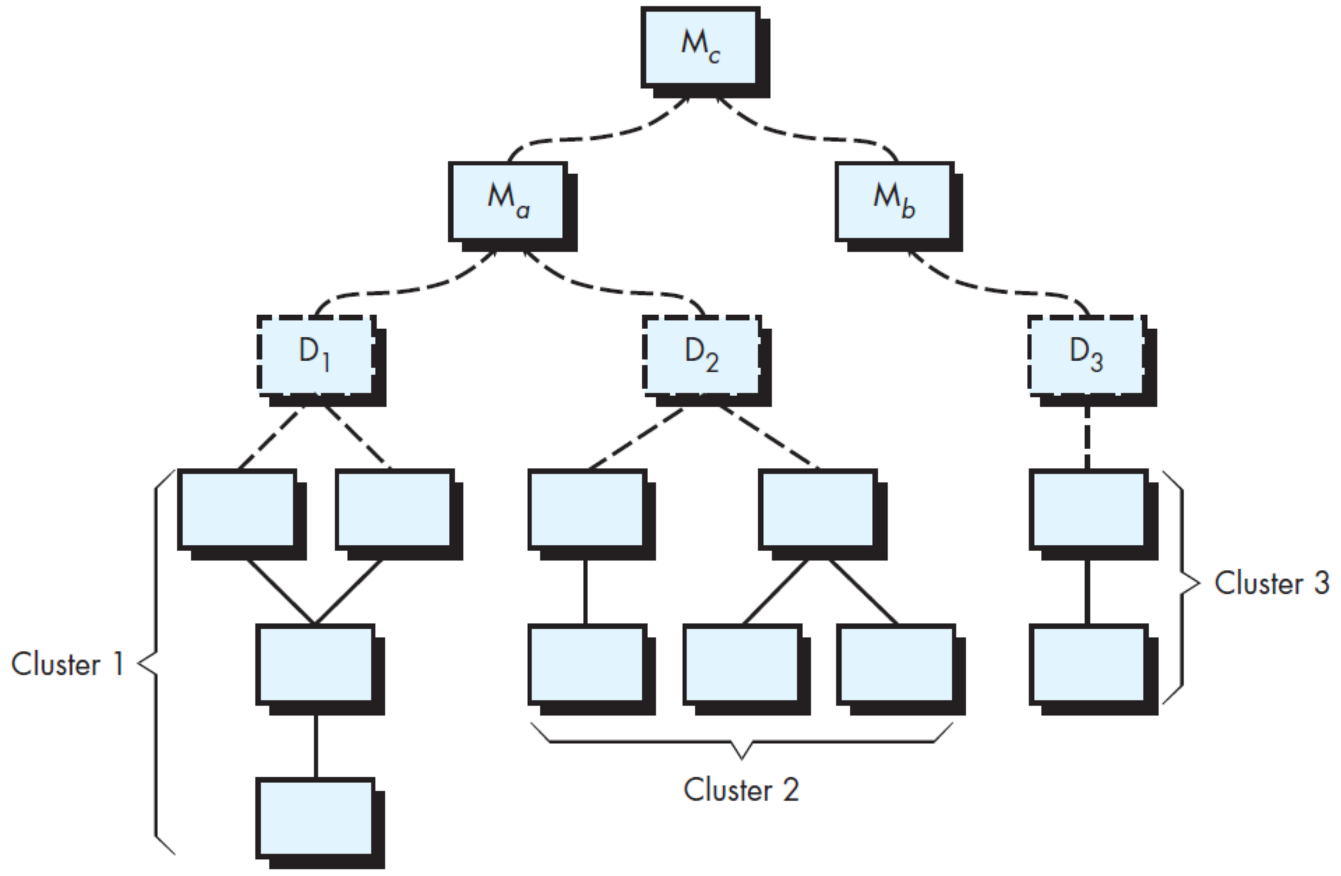
- Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing
- The objective is to take unit-tested components and build a program structure that has been dictated by design

Top down Integration

Depth first vs breadth first integration



Bottom Up Integration



Regression Testing

- Each time a new module is added as part of integration testing, the software changes
- New data flow paths are established, new I/O may occur, and new control logic is invoked
- Side effects associated with these changes may cause problems with functions that previously worked flawlessly
- In the context of an integration test strategy, regression testing is the reexecution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects
- Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors

Regression Testing

- The *regression test suite* (the subset of tests to be executed) contains three different classes of test cases:
  - – A representative sample of tests that will exercise all software functions.
  - – Additional tests that focus on software functions that are likely to be affected by the change
  - – Tests that focus on the software components that have been changed

Smoke Testing

- Smoke testing is an integration testing approach that is commonly used when product software is developed
- It encompasses the following activities:
  - Software components that have been translated into code are integrated into a build
  - A series of tests is designed to expose errors that will keep the build from properly performing its function
  - The build is integrated with other builds, and the entire product (in its current form) is smoke tested daily
- The integration approach may be top down or bottom up

Benefits of Smoke Tests

- *Integration risk is minimized*
  - Because smoke tests are conducted daily, incompatibilities and other show-stopper errors are uncovered early, thereby reducing the likelihood of serious schedule impact when errors are uncovered
- *The quality of the end product is improved*
  - Because the approach is construction (integration) oriented, smoke testing is likely to uncover functional errors as well as architectural and component-level design errors. If these errors are corrected early, better product quality will result.
- *Error diagnosis and correction are simplified*
  - Like all integration testing approaches, errors uncovered during smoke testing are likely to be associated with “new software increments”
- *Progress is easier to assess*
  - With each passing day, more of the software has been integrated and more has been demonstrated to work. This improves team morale and gives managers a good indication that progress is being made.

Test strategies for web apps

- The content model for the WebApp is reviewed to uncover errors
- The interface model is reviewed to ensure that all use cases can be accommodated.
- The design model for the WebApp is reviewed to uncover navigation errors.
- UI Testing
- Each functional component is unit tested.
- Navigation throughout the architecture is tested.
- The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration
- Security testing
- Performance testing
- The WebApp is tested by a controlled and monitored population of end users. The results of their interaction with the system are evaluated for errors

Test strategies for mobile apps

- User-experience testing
- Device compatibility testing
- Performance testing. Testers check nonfunctional requirements unique to mobile devices (e.g., download times, processor speed, storage capacity, power availability).
- Connectivity testing.
- Security testing
- Testing-in-the-wild . The app is tested under realistic conditions on actual user devices in a variety of networking environments around the globe.
- Certification testing. Testers ensure that the MobileApp meets the standards established by the app stores that will distribute it.

Validation Testing

- At the validation level, the distinction between different software categories disappears
- Testing focuses on user-visible actions and user-recognizable output from the system
- Validation Criteria along with user stories
- Alpha-beta testing
  - Customer acceptance Testing

Discussion

- Who should perform the validation test—the software developer or the software user?
- Justify your answer

System Testing

Recovery Testing

Security Testing

Stress Testing

Performance Testing

Deployment Testing

Discussion

- Identify a few non-functional tests to be performed for your project applications
  - Identify types of non-functional tests
  - Identify a few tests within those types

Summary

- The objective of software testing is to uncover errors
- For conventional software, this objective is achieved through a series of test steps. Unit and integration tests concentrate on functional verification of a component and incorporation of components into the software architecture
- Validation testing demonstrates traceability to software requirements, and system testing validates software once it has been incorporated into a larger system
- Each test step is accomplished through a series of systematic test techniques that assist in the design of test cases
- With each testing step, the level of abstraction with which software is considered is broadened

Testing conventional/web/mobile applications

Pressman R S, Bruce R.Maxim, “Software engineering - A Practitioner’s Approach”, Eighth Edition, Tata McGraw-Hill, 2014
Chapter 23, 25, 26

Topics Covered

Topic	**Text Book Reference**
Testing conventional applications	**23.1, 23.2, 23.3, 23.6**
Testing Web Apps	**25.1, 25.2, 25.3, 25.4, 25.7, 25.8, 25.9,**
Testing Mobile Apps	**26.1, 26.2, 26.3**

Testing fundamentals

- The goal of testing is to find errors, and a good test is one that has a high probability of finding an error
- Therefore, you should design and implement a computer-based system or a product with “testability” in mind
- At the same time, the tests themselves must exhibit a set of characteristics that achieve the goal of finding the most errors with a minimum of effort
- Testability - “ Software testability is simply how easily [a computer program] can be tested.”

Testability Characteristics

Testability	*Operability*	"The better it works, the more efficiently it can be tested."
	Observability	"What you see is what you test."
	Controllability	"The better we can control the software, the more the testing can be automated and optimized."
	Decomposability	"By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting."
	Simplicity	"The less there is to test, the more quickly we can test it."
	Stability	"The fewer the changes, the fewer the disruptions to testing."
	Understandability	"The more information we have, the smarter we will test."

Test characteristics

- *A good test has a high probability of finding an error*
- *A good test is not redundant*
- *A good test should be “best of breed”*
- *A good test should be neither too simple nor too complex*

Internal and external views of testing

Black-box testing

- allowed to tests that are conducted at the software interface.
- A black-box test examines some fundamental aspect of a system with little regard for the internal logical structure of the software

White-box testing

- of software is predicated on close examination of procedural detail
- Logical paths through the software and collaborations between components are tested by exercising specific sets of conditions and/or loops

White box testing

- White-box testing, sometimes called glass-box testing or structural testing, is a test-case design philosophy that uses the control structure described as part of component-level design to derive test cases
- Using white-box testing methods, you can derive test cases that
  - (1) guarantee that all independent paths within a module have been exercised at least once,
  - (2) exercise all logical decisions on their true and false sides,
  - (3) execute all loops at their boundaries and within their operational bounds, and
  - (4) exercise internal data structures to ensure their validity.

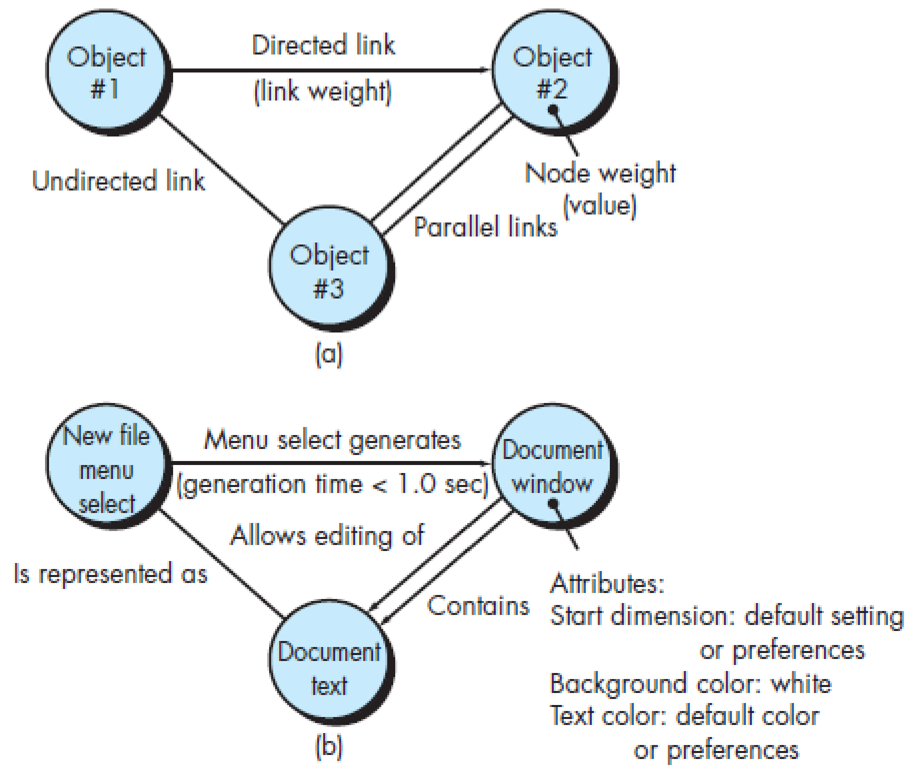
Black box testing

- *Black-box testing* , also called *behavioral testing* or *functional testing,* focuses on the functional requirements of the software
- That is, black-box testing techniques enable you to derive sets of input conditions that will fully exercise all functional
- requirements for a program
- Black-box testing is not an alternative to white-box techniques
- Rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods.

Black box testing

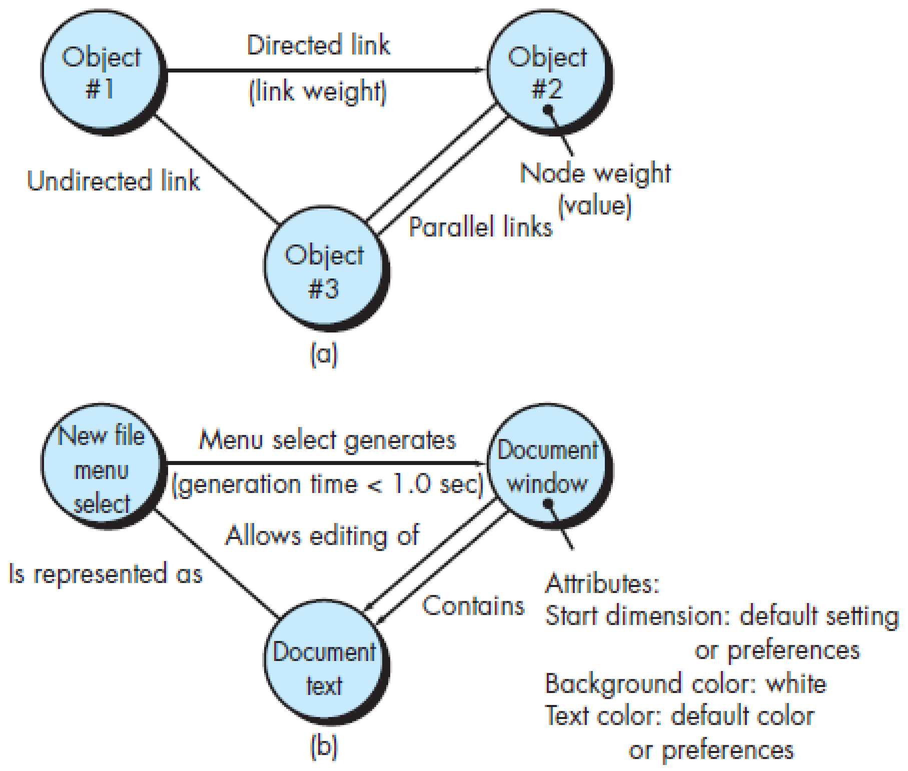
- Black-box testing attempts to find errors in the following categories:
  - (1) incorrect or missing functions,
  - (2) interface errors,
  - (3) errors in data structures or external database access,
  - (4) behavior or performance errors, and
  - (5) initialization and termination errors.
- Unlike white-box testing, which is performed early in the testing process, blackbox testing tends to be applied during later stages of testing

Graph-Based Testing Methods



- The first step in black-box testing is to understand the objects that are modeled in software and the relationships that connect these objects
- Once this has been accomplished, the next step is to define a series of tests that verify "all objects have the expected relationship to one another"

Graph-Based Testing Methods



- You can then derive test cases by traversing the graph and covering each of the relationships shown
- These test cases are designed in an attempt to find errors in any of the relationships

Discussion

Model any of the scenarios in your project and derive graph-based tests for the same.

Equivalence Partitioning

- Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived
- An ideal test case single-handedly uncovers a class of errors (e.g., incorrect processing of all character data) that might otherwise require many test cases to be executed before the general error is observed
- An equivalence class represents a set of valid or invalid states for input conditions

Boundary value analysis

- A greater number of errors occurs at the boundaries of the input domain rather than in the “center.”
- It is for this reason that *boundary value analysis* (BVA) has been developed as a testing technique
- Boundary value analysis leads to a selection of test cases that exercise bounding values
- Boundary value analysis is a test-case design technique that complements equivalence partitioning
- Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the “edges” of the class

Summary

- The primary objective for test-case design is to derive a set of tests that have the highest likelihood for uncovering errors in software
- To accomplish this objective, two different categories of test-case design techniques are used: white-box testing and black-box testing
- White-box tests focus on the program control structure. Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been exercised\
- Black-box tests are designed to validate functional requirements without regard to the internal workings of a program

Chapter 25, **25.1, 25.2, 25.3, 25.4, 25.7, 25.8, 25.9**

TESTING WEB APPLICATIONS

Dimensions of Quality

- Content
- Function
- Structure
- Usability
- Navigability
- Performance
- Compatibility
- Interoperability
- Security

Errors within a WebApp Environment

1. Because many types of WebApp tests uncover problems that are first evidenced on the client side, you often see a symptom of the error, not the error itself
2. Because a WebApp is implemented in a number of different configurations and within different environments, it may be difficult or impossible to reproduce an error outside the environment in which the error was originally encountered
3. Although some errors are the result of incorrect design or improper HTML (or other programming language) coding, many errors can be traced to the WebApp configuration
4. Because WebApps reside within a client-server architecture, errors can be difficult to trace across three architectural layers: the client, the server, or the network itself
5. Some errors are due to the static operating environment (i.e., the specific configuration in which testing is conducted), while others are attributable to the dynamic operating environment

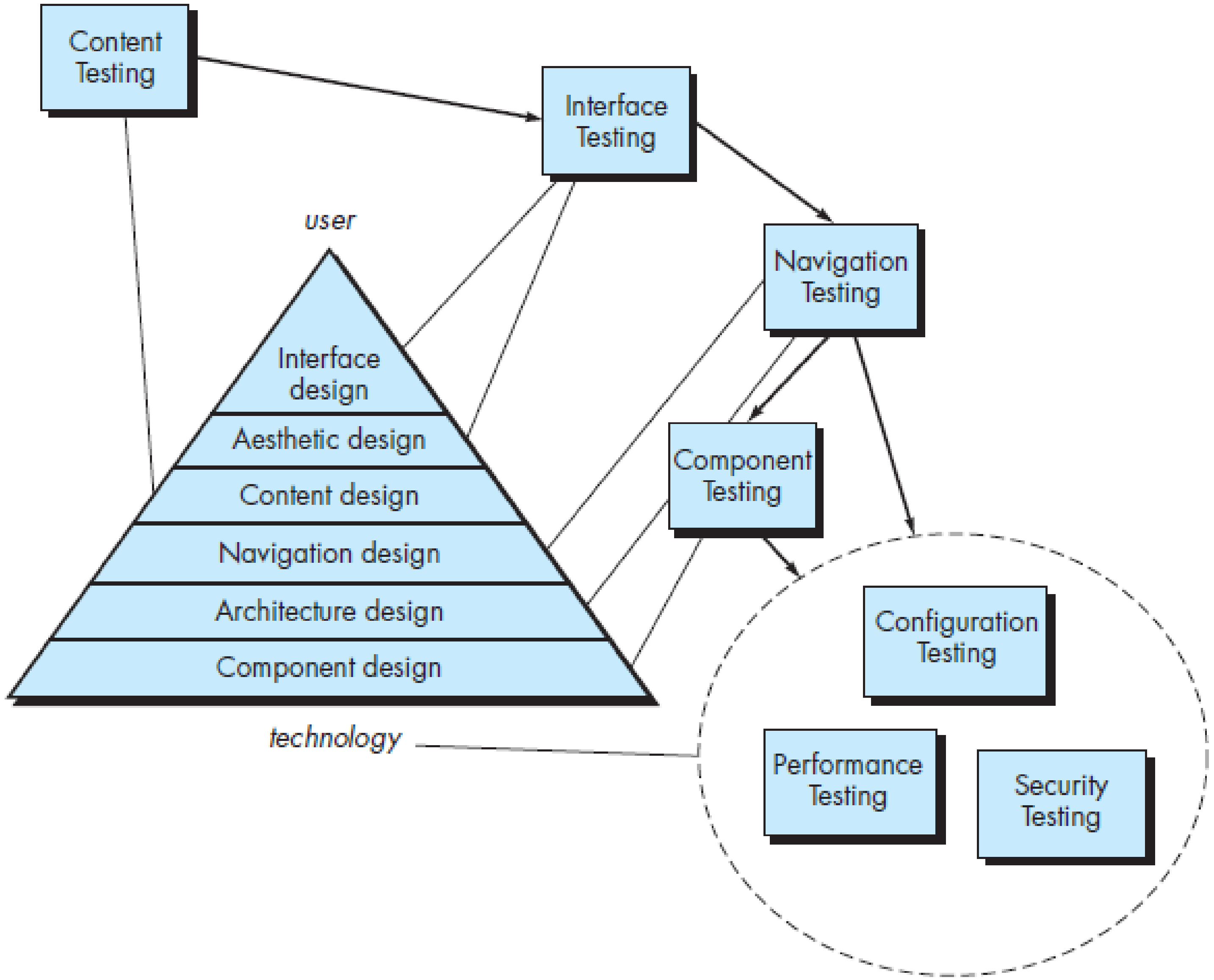
Testing Strategy

- The content model for the WebApp is reviewed to uncover errors
- The interface model is reviewed to ensure that all use cases can be accommodated
- The design model for the WebApp is reviewed to uncover navigation errors
- The user interface is tested to uncover errors in presentation and/or navigation mechanics
- Selected functional components are unit tested
- Navigation throughout the architecture is tested
- The WebApp is implemented in a variety of different environmental configurations and is tested for compatibility with each configuration
- Security tests are conducted in an attempt to exploit vulnerabilities in the WebApp or within its environment
- Performance tests are conducted
- The WebApp is tested by a controlled and monitored population of end users; the results of their interaction with the system are evaluated for content and navigation errors, usability concerns, compatibility concerns, and WebApp security, reliability, and performance.

Test Planning

- A WebApp test plan identifies
  - the task set to be applied as testing commences,
  - the work products to be produced as each testing task is executed, and
  - the manner in which the results of testing are evaluated, recorded, and reused when regression testing is conducted

The testing Process



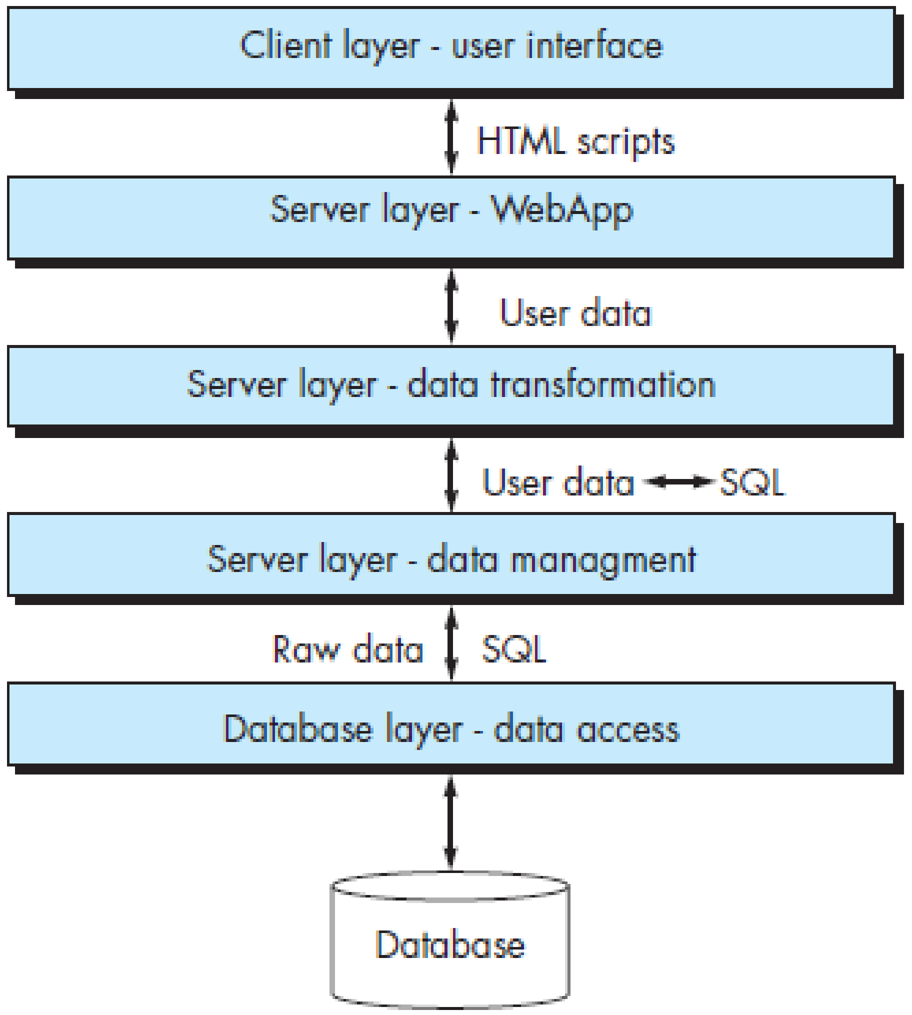
Content Testing

- Content testing has three important objectives:
  - to uncover syntactic errors (e.g., typos, grammar mistakes) in text-based documents, graphical representations, and other media;
  - to uncover semantic errors (i.e., errors in the accuracy or completeness of information) in any content object presented as navigation occurs, and
  - to find errors in the organization or structure of content that is presented to the end user

Content Testing

Database Testing

- WebApps interface with sophisticated database management systems and build dynamic content objects that are created in real time using the data acquired from a database
- For example, a financial services WebApp can produce complex text-based, tabular, and graphical information about a specific equity (e.g., a stock or mutual fund)
- The composite content object that presents this information is created dynamically after the user has made a request for information about a specific equity

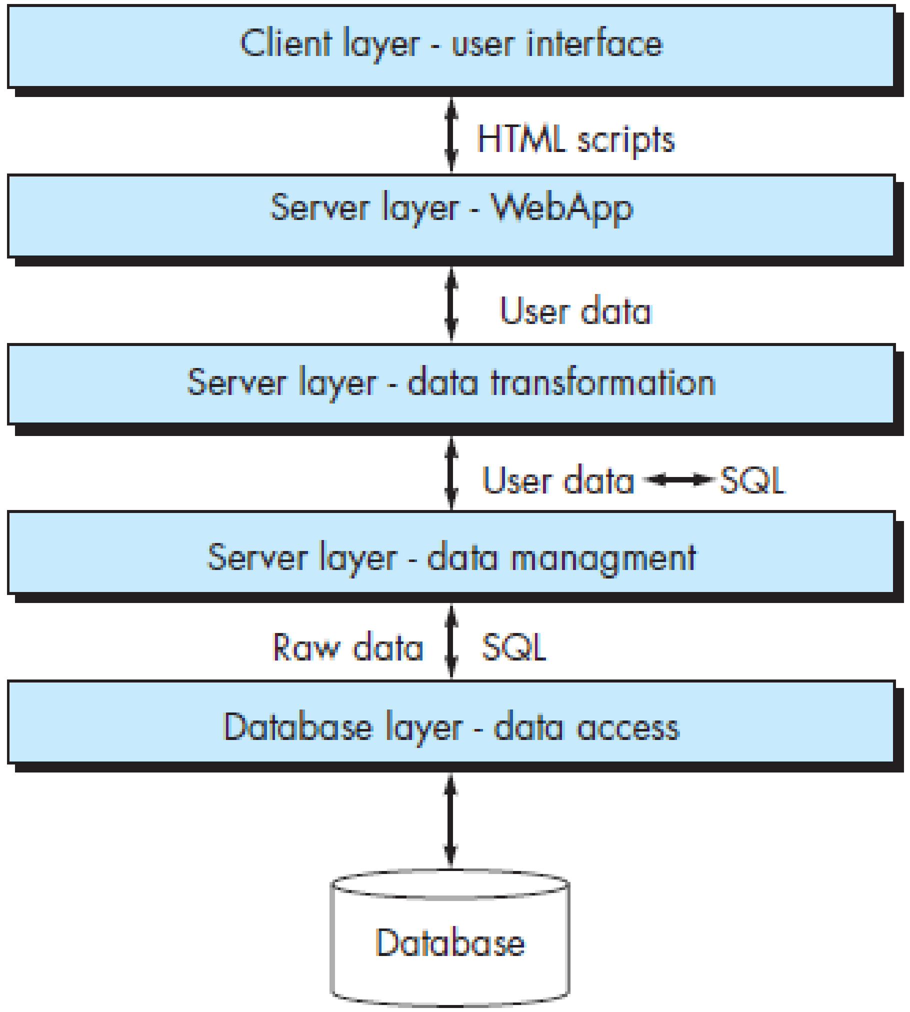


Client layer - user interface
HTML scripts
Server layer - WebApp
User data
Server layer - data transformation
User data ↔ SQL
Server layer - data managment
Raw data
SQL
Database layer - data access
Database

Content Testing

Database Testing

1. an equities database is queried,
2. relevant data are extracted from the database,
3. the extracted data must be organized as a content object, and
4. this content object (representing customized information requested by an end user) is transmitted to the client environment for display


Client layer - user interface
HTML scripts
Server layer - WebApp
User data
Server layer - data transformation
User data ⟷ SQL
Server layer - data managment
Raw data
SQL
Database layer - data access
Database

Content Testing

Database Testing – Complicating factors

- *The original client-side request for information is rarely presented in the form [e.g. SQL] that can be input to a database management system (DBMS)*
  - Therefore, tests should be designed to uncover errors made in translating the user's request into a form that can be processed by these DBMS
- *The database may be remote to the server that houses the WebApp*
  - Therefore, tests that uncover errors in communication between the WebApp and the remote database must be developed

Content Testing

Database Testing – Complicating factors

- *Raw data acquired from the database must be transmitted to the WebApp server and properly formatted for subsequent transmittal to the client*
  - Therefore, tests that demonstrate the validity of the raw data received by the WebApp server must be developed, and additional tests that demonstrate the validity of the transformations applied to the raw data to create valid content objects must also be created
- *The dynamic content object(s) must be transmitted to the client in a form that can be displayed to the end user*
  - Therefore, a series of tests must be designed to (1) uncover errors in the content object format and (2) test compatibility with different client environment configurations

User Interface Testing

- Verification and validation of a WebApp user interface occurs at three distinct points
- During requirements analysis, the interface model is reviewed to ensure that it conforms to stakeholder requirements and to other elements of the requirements model
- During design the interface design model is reviewed to ensure that generic quality criteria established for all user interfaces have been achieved and that application-specific interface design issues have been properly addressed
- During testing, the focus shifts to the execution of application-specific aspects of user interaction as they are manifested by interface syntax and semantics. In addition, testing provides a final assessment of usability

Interface Testing Strategy

- *Interface testing* exercises interaction mechanisms and validates aesthetic aspects of the user interface
- The overall strategy for interface testing is to
  - Uncover errors related to specific interface mechanisms (e.g., errors in the proper execution of a menu link or the way data are entered in a form) and
  - Uncover errors in the way the interface implements the semantics of navigation, WebApp functionality, or content display

Testing Interface Mechanisms

- Links
- Forms
- Client side scripting
- Dynamic HTML
- Pop up windows
- Streaming content
- Cookies
- Application specific interface mechanisms

Usability Testing

- Usability testing can occur at a variety of different levels of abstraction:
  - the usability of a specific interface mechanism (e.g., a form) can be assessed,
  - the usability of a complete Web page (encompassing interface mechanisms, data objects, and related functions) can be evaluated, or
  - the usability of the complete WebApp can be considered

Usability Testing

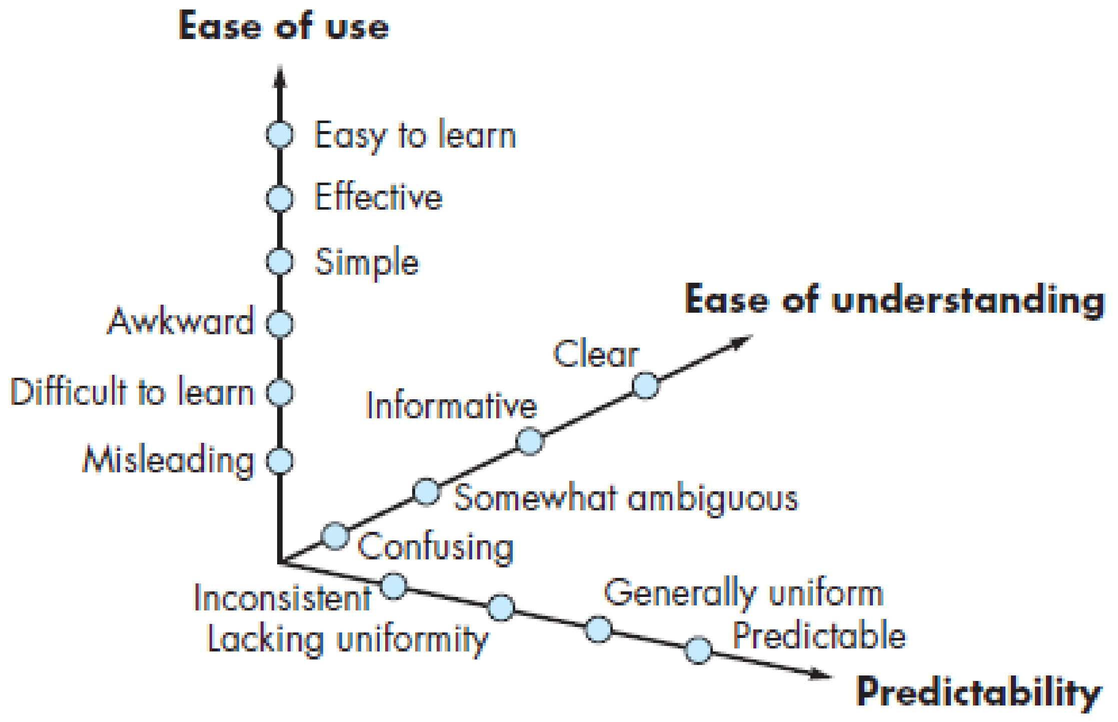
Usability categories

- Interactivity
- Layout
- Readability
- Aesthetics
- Display characteristics
- Time sensitivity
- Personalization
- Accessibility

Usability Testing

Qualitative assessment of usability

- Figure illustrates possible set of assessment “grades” that can be selected by users
- These grades are applied to each feature individually, to a complete Web page, or to the WebApp as a whole



Compatibility Tests

- Different computers, display devices, operating systems, browsers, and network connection speeds can have a significant influence on WebApp operation
- Each computing configuration can result in differences in client-side processing speeds, display resolution, and connection speeds
- Operating system vagaries may cause WebApp processing issues
- Different browsers sometimes produce slightly different results, regardless of the degree of HTML standardization within the WebApp
- Required plug-ins may or may not be readily available for a particular configuration
- *Compatibility testing* strives to uncover these problems before the WebApp goes online

Compatibility Tests

- The first step in compatibility testing is to define a set of “commonly encountered” client-side computing configurations and their variants
- In essence, a tree structure is created, identifying each computing platform, typical display devices, the operating systems supported on the platform, the browsers available, likely Internet connection speeds, and similar information
- Next, a series of compatibility validation tests are derived, often adapted from existing interface tests, navigation tests, performance tests, and security tests
- The intent of these tests is to uncover errors or execution problems that can be traced to configuration differences

Discussion

Compatibility is an important quality dimension. What must be tested to ensure that compatibility exists for a WebApp?

25.7

CONFIGURATION TESTING

Configuration Testing

- Configuration variability and instability are important factors that make WebApp testing a challenge
- Hardware, OS, browsers, storage capacity etc are difficult to predict for each user
- The result can be a client-side environment that is prone to errors that are both subtle and significant
- One user's impression of the WebApp and the manner in which she interacts with it can differ significantly from another user's experience, if both users are not working within the same client-side configuration
- The job of *configuration testing* is not to exercise every possible client-side configuration
- Rather, it is to test a set of probable client-side and server-side configurations to ensure that the user experience will be the same on all of them and to isolate errors that may be specific to a particular configuration

Discussion

Which errors tend to be more serious—client-side errors or server-side errors? Why?

25.8

SECURITY TESTING

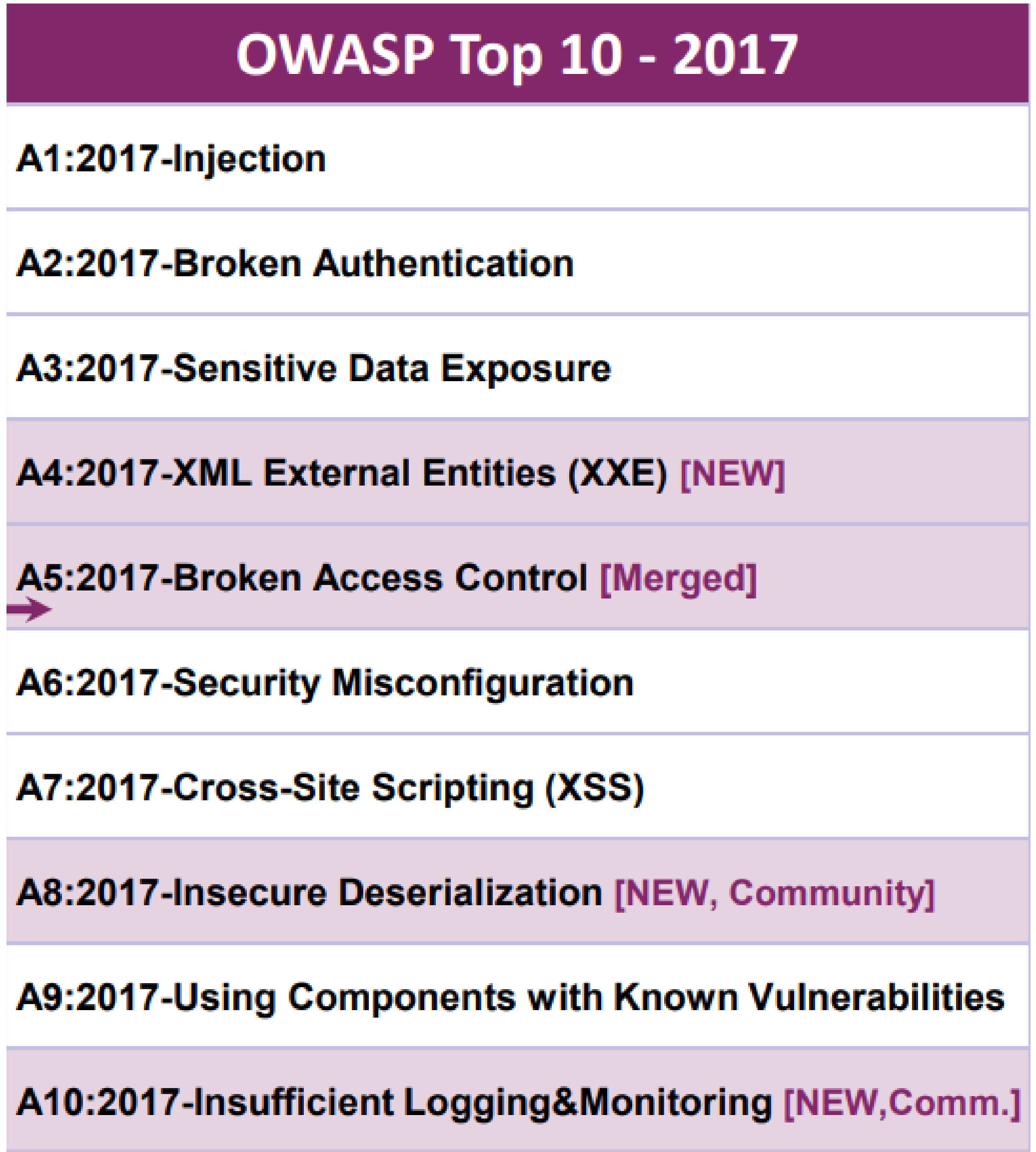
Security Testing

- WebApp security is a complex subject that must be fully understood before effective security testing can be accomplished
- WebApps and the client-side and server-side environments in which they are housed represent an attractive target for external hackers, disgruntled employees, dishonest competitors, and anyone else who wishes to steal sensitive information, maliciously modify content, degrade performance, disable functionality, or embarrass a person, organization, or business

Security Testing

- Security tests are designed to probe vulnerabilities of the client-side environment, the network communications that occur as data are passed from client to server and back again, and the server-side environment
- Each of these domains can be attacked, and it is the job of the security tester to uncover weaknesses that can be exploited by those with the intent to do so
- On the client side, vulnerabilities can often be traced to preexisting bugs in browsers, e-mail programs, or communication software
- On the server side, vulnerabilities include denial-of-service attacks and malicious scripts that can be passed along to the client side or used to disable server operations
- In addition, server-side databases can be accessed without authorization (data theft)

OWASP Top 10 attacks



OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE) [NEW]
A5:2017-Broken Access Control [Merged]
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization [NEW, Community]
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Not from Text book
But extremely useful to understand

https://www.veracode.com/directory/owasp-top-10

OWASP Top 10 attacks

Three of them from security testing perspective

Injection

- Injection flaws, such as SQL injection, LDAP injection, and CRLF injection, occur when an attacker sends untrusted data to an interpreter that is executed as a command without proper authorization.
- **Application security testing** can easily detect injection flaws. Developers should use parameterized queries when coding to prevent injection flaws

Broken Access Control

- Improperly configured or missing restrictions on authenticated users allow them to access unauthorized functionality or data, such as accessing other users' accounts, viewing sensitive documents, and modifying data and access rights.
- **Penetration testing** is essential for detecting non-functional access controls; other testing methods only detect where access controls are missing

Cross-Site Scripting

- Cross-site scripting (XSS) flaws give attackers the capability to inject client-side scripts into the application, for example, to redirect users to malicious websites.
- **Developer training complements security testing** to help programmers prevent cross-site scripting with best coding best practices, such as encoding data and input validation**.**

25.9

PERFORMANCE TESTING

Performance Testing

- *Performance testing* is used to uncover performance problems that can result from a lack of server-side resources, inappropriate network bandwidth, inadequate database capabilities, faulty or weak operating system capabilities, poorly designed WebApp functionality, and other hardware or software issues that can lead to degraded client-server performance
- The intent is twofold:
  - to understand how the system responds as *loading* (i.e., number of users, number of transactions, or overall data volume), and
  - to collect metrics that will lead to design modifications to improve performance

Performance Testing

Questions to be answered

- Does the server response time degrade to a point where it is noticeable and unacceptable?
- At what point (in terms of users, transactions, or data loading) does performance become unacceptable?
- What system components are responsible for performance degradation?
- What is the average response time for users under a variety of loading conditions?
- Does performance degradation have an impact on system security?
- Is WebApp reliability or accuracy affected as the load on the system grows?
- What happens when loads that are greater than maximum server capacity are applied?
- Does performance degradation have an impact on company revenues?

Load Testing

- The intent of load testing is to determine how the WebApp and its server-side environment will respond to various loading conditions
- As testing proceeds, permutations to the following variables define a set of test conditions:
  - *N,* number of concurrent users
  - *T,* number of online transactions per unit of time
  - *D,* data load processed by the server per transaction

Load Testing

- The intent of load testing is to determine how the WebApp and its server-side environment will respond to various loading conditions
- As testing proceeds, permutations to the following variables define a set of test conditions:
  - *N,* number of concurrent users
  - *T,* number of online transactions per unit of time
  - *D,* data load processed by the server per transaction
- You should examine these measures to determine whether a decrease in performance can be traced to a specific combination of *N, T,* and *D*

Load Testing

- Load testing can also be used to assess recommended connection speeds for users of the WebApp
- Overall throughput, *P,* is computed in the following manner:

$P = N * T * D$

- As an example, consider a popular sports news site. At a given moment, 20,000 concurrent users submit a request (a transaction, *T*) once every 2 minutes on average. Each transaction requires the WebApp to download a new article that averages 3K bytes in length. Therefore, throughput can be calculated as:

P = [20,000 * 0.5 * 3Kb]/60 = 500 Kbytes/sec

= 4 megabits per second

- The network connection for the server would therefore have to support this data rate and should be tested to ensure that it does

Stress Testing

- *Stress testing* is a continuation of load testing, but in this instance the variables, *N, T,* and *D* are forced to meet and then exceed operational limits
- The intent of these tests is to answer each of the following questions:
  - Does the system degrade “gently,” or does the server shut down as capacity is exceeded?
  - Does server software generate “server not available” messages? More generally, are users aware that they cannot reach the server?
  - Does the server queue resource requests and empty the queue once capacity demands diminish?
  - Are transactions lost as capacity is exceeded?
  - …..

Summary

- The goal of WebApp testing is to exercise each of the many dimensions of WebApp quality with the intent of finding errors or uncovering issues that may lead to quality failures
- Testing focuses on content, function, structure, usability, navigability, performance, compatibility, interoperability, capacity, and security

Chapter 26, **26.1, 26.2, 26.3**

TESTING MOBILE APPLICATIONS

Mobile App Testing

- There are several important questions to ask when creating a MobileApp testing strategy
  - Do you have to build a fully functional prototype before you test with users?
  - Should you test with the user’s device or provide a device for testing?
  - What devices and user groups should you include in testing?
  - What are the benefits/drawbacks of lab testing versus remote testing?

Testing Guidelines

- Understand the network and device landscape before testing to identify bottlenecks
- Conduct tests in uncontrolled real-world test conditions
- Select the right automation test tool
- Use the Weighted Device Platform Matrix method to identify the most critical hardware/platform combination to test
- Check the end-to-end functional flow in all possible platforms at least once
- Conduct performance testing, GUI testing, and compatibility testing using actual devices
- Measure performance only in realistic conditions of wireless traffic and user load

MobileApp Testing - Checklist

- Conceptual
- Unit and System
- Ux
- Stability
- Connectivity
- Performance
- Device Compatibility
- Security
- Certification

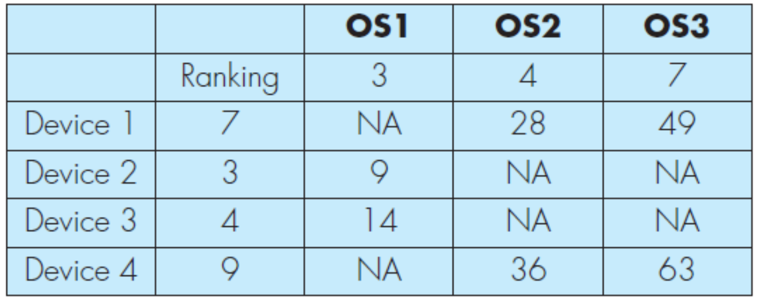
Building a test matrix

- A weighted device platform matrix (WDPM) helps ensure that test coverage includes each combination of mobile device and context variables
- The WDPM can also be used to help prioritize the device/context combinations so that the most important are tested first

		OS1	OS2	OS3
	Ranking	3	4	7
Device 1	7	NA	28	49
Device 2	3	9	NA	NA
Device 3	4	14	NA	NA
Device 4	9	NA	36	63

Building a test matrix

1. list the important operating system variants as the matrix column labels
2. list the targeted devices as the matrix row labels
3. assign a ranking (e.g., 0 to 10) to indicate the relative importance of each operating system and each device, and
4. compute the product of each pair of rankings and enter each product as the cell entry in the matrix



		OS1	**OS2**	**OS3**
	Ranking	3	4	7
Device 1	7	NA	28	49
Device 2	3	9	NA	NA
Device 3	4	14	NA	NA
Device 4	9	NA	36	63

Stress Testing

- *Stress testing* for mobile apps attempts to find errors that will occur under extreme operating conditions
- In addition, it provides a mechanism for determining whether the MobileApp will degrade gracefully without compromising security
- Among the many actions that might create extreme conditions are:
  - running several mobile apps on the same device,
  - infecting system software with viruses or malware,
  - attempting to take over a device and use it to spread spam,
  - forcing the mobile app to process inordinately large numbers of transactions, and
  - storing inordinately large quantities of data on the device

Testing in a Production Environment

- Many MobileApp developers advocate testing-in-the-wild , or testing in the users' native environments with the production release versions of the MobileApp resources
- Some of the characteristics of in-the-wild testing include adverse and unpredictable environments, outdated browsers and plug-ins, unique hardware, and imperfect connectivity
- Creating test environments in-house is an expensive and error-prone process
- Cloud-based testing can offer a standardized infrastructure and preconfigured software images, freeing the MobileApp team from the need to worry about finding servers or purchasing their own licenses for software and testing tools

CONSIDERING THE SPECTRUM OF USER INTERACTION

- Gesture Testing
- Voice Input and Recognition
- Virtual Key Board Input
- Alerts and Extraordinary Conditions

Summary

- The goal of MobileApp testing is to exercise each of the many dimensions of MobileApp quality with the intent of finding errors or uncovering issues that may lead to quality failures
- Testing focuses on quality elements such as content, function, structure, usability, use of context, navigability, performance, power management, compatibility, interoperability, capacity, and security
- It incorporates reviews and usability assessments that occur as the MobileApp is designed, and tests that are conducted once the MobileApp has been implemented and deployed on an actual device