投票 最多可选1项

设置

Are you ready ?

A Yes

B No

提交

Review

- ✓ Concept: "Good" testing, test oracle(expected result)
- ✓ **White-box testing**
  - statement coverage, branch coverage, decision coverage
  - branch/decision coverage, combination coverage, basic path coverage
  - Data flow testing, loop testing
- ✓ **Black-box testing**
  - Equivalence Partitioning, boundary value analysis
  - Cause-effect graph & Decision table
- ✓ Combination testing
- ✓ Usage-based testing (Operational Profile)

Software Engineering

Part 3
Quality Management

Chapter 24
Testing Object-Oriented Applications

Contents

- 24.1 Broadening the View of Testing
- 24.2 Testing OOA and OOD Models
  - 24.2.1 Correctness of OOA and OOD Models
  - 24.2.2 Consistency of Object-Oriented Models
- 24.3 Object-Oriented Testing Methods
  - 24.3.1 Unit Testing in the OO Context
  - 24.3.2 Integration Testing in the OO Context
  - 24.3.3 Validation Testing in an OO Context
- 24.4 Object-Oriented Testing Methods
  - 24.4.1 The Test-Case Design Implications of OO Concepts
  - 24.4.2 Applicability of Conventional Test-Case Design Methods
  - 24.4.3 Fault-Based Testing
  - 24.4.4 Scenario-Based Test Design

Contents

- 24.5 Testing Methods Applicable at the Class level
  - 24.5.1 Random Testing for OO Classes
  - 24.5.2 Partition Testing at the Class Level
- 24.6 Interclass Test-Case Design
  - 24.6.1 Multiple Class Testing
  - 24.6.2 Tests Derived from Behavior Models

24.1 Broadening the View of Testing

Three things must be done for OO Testing:

- the strategy for unit and integration testing must change significantly
- error discovery techniques applied to object-oriented analysis and design models
- the design of test cases must account for the unique characteristics of OO software.

24.2 ‘Testing’ OO Models

Review of OO analysis and design models (e.g., classes, attributes, operations, messages appear at the analysis, design, and code level)

Example:

A class in which a number of attributes are defined during the first iteration of analysis. An extraneous attribute and two operations are appended to the class (due to a misunderstanding of the problem domain)

Q1: What problems and unnecessary effort will be costed during analysis?
(Generating subclass? Class relationship? Behavior characterizing?)

Q2: What problems and unnecessary effort will be costed during design?
(Improper allocation of class? Unnecessary design work? Messaging model?)

Q3: What problems and unnecessary effort will be costed during coding?
(Generating code, testing, modification)

24.2.1 Correctness of OO Models

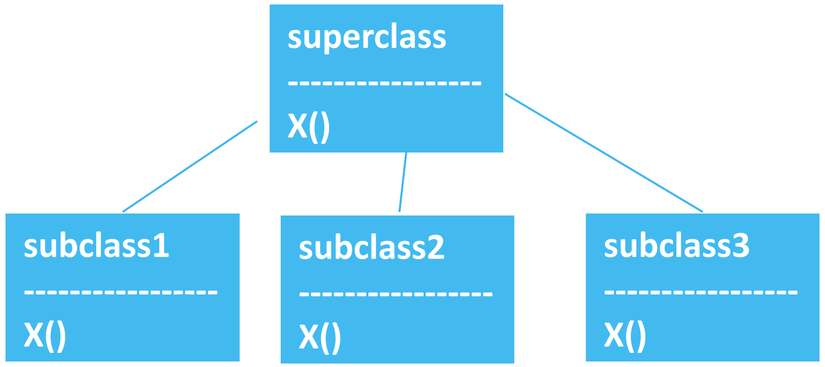
- During analysis and design, semantic correctness can be asesssed based on the model's conformance to the real world problem domain.

- Semantically correct : If the model accurately reflects the real world (to a level of detail that is appropriate to the stage of development at which the model is reviewed).
  - Presented to problem domain experts who will examine the class definitions and hierarchy for omissions and ambiguity.
  - Class relationships (instance connections) : Whether they accurately reflect real-world object connections.

24.3 OO Testing Strategies

- Procedural software
  - unit = single program, function, or procedure
- Object oriented software
  - unit = class
  - unit testing = **intra-class testing**
  - integration testing = **inter-class testing**
    - cluster of classes
  - dealing with single methods separately is usually too expensive (complex scaffolding), so methods are usually tested in the context of the class they belong to

1. No hierarchical control structure, top-down, bottom-up can not be used.
2. Direct and indirect interactions of class, incremental integration can not be used.



Testing: X()

test x() in the context of each of the subclass and superclass

24.3 OO Testing Strategies

- Unit testing
  - the concept of the unit changes
  - the smallest testable unit is the encapsulated class
  - a single operation can no longer be tested in isolation (the conventional view of unit testing) but rather, as part of a class

- Integration Testing
  - *Thread-based testing* integrates the set of classes required to respond to one input or event for the system
  - *Use-based testing* begins the construction of the system by testing those classes (called *independent classes*) that use very few (if any) of server classes. After the independent classes are tested, the next layer of classes, called *dependent classes*
  - *Cluster testing* defines a cluster of collaborating classes (determined by examining the CRC and object-relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

24.4 OOT Methods

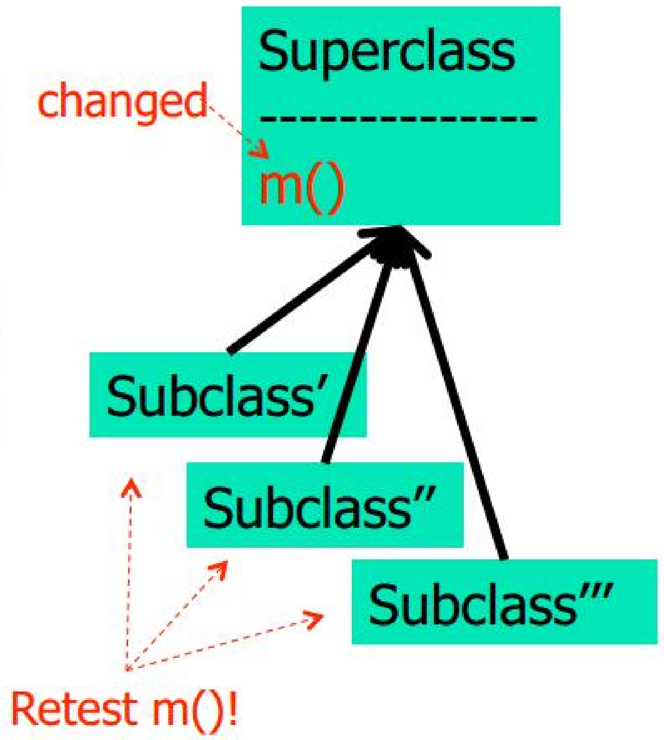
Account class:
open, setup
(initial operation)

Berard [Ber93] proposes the following approach:

1. Each test case should be uniquely identified and should be explicitly associated with the class to be tested
2. The purpose of the test should be stated
3. A list of testing steps should be developed for each test:
    a. a list of specified states for the object that is to be tested
    b. a list of messages and operations that will be exercised as a consequence of the test
    c. a list of **exceptions** that may occur as the object is tested
    d. a list of external conditions (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)
    e. supplementary information that will aid in understanding or implementing the test.

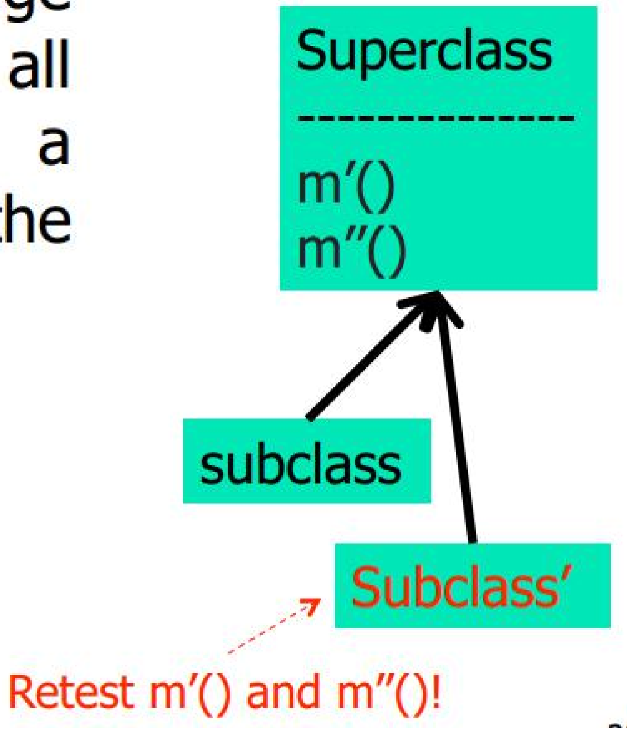
24.4 Testing of Inheritance (1)

- **<u>Principle:</u>** inherited methods should be retested in the context of a subclass
  - **Example 1**: if we change some method m() in a superclass, we need to retest m() inside all subclasses that inherit it



24.4 Testing of Inheritance (2)

- **Example 2**: if we add or change a subclass, we need to retest all methods inherited from a superclass in the context of the new/changed subclass



24.4 Testing Methods

- Fault-based testing

  The tester looks for plausible faults (i.e., aspects of the implementation of the system that may result in defects). To determine whether these faults exist, test cases are designed to exercise the design or code.

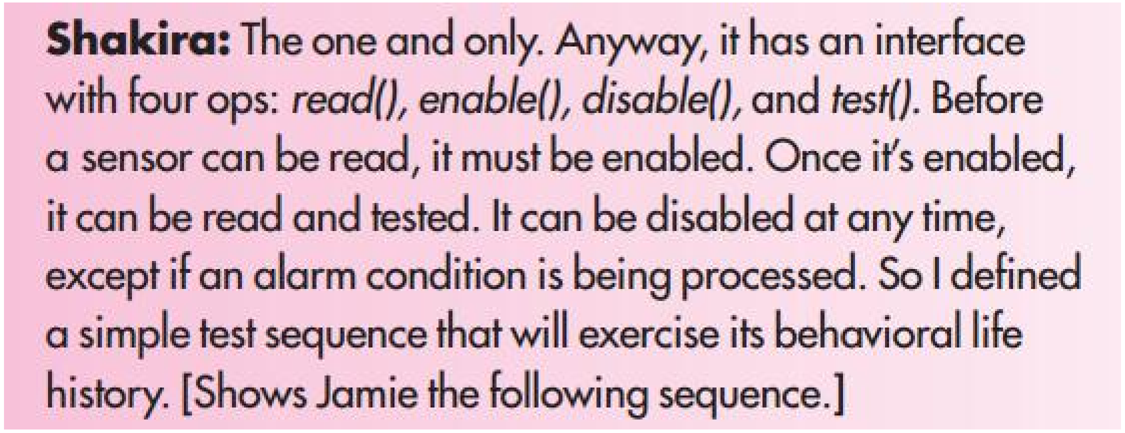
- Class Testing and the Class Hierarchy

  Inheritance does not obviate the need for thorough testing of all derived classes. In fact, it can actually complicate the testing process.

- Scenario-Based Test Design

  Scenario-based testing concentrates on what the user does, not what the product does. This means capturing the tasks (via use-cases) that the user has to perform, then applying them and their variants as tests.

24.4 Testing Methods


Shakira: The one and only. Anyway, it has an interface with four ops: *read(), enable(), disable(),* and *test().* Before a sensor can be read, it must be enabled. Once it's enabled, it can be read and tested. It can be disabled at any time, except if an alarm condition is being processed. So I defined a simple test sequence that will exercise its behavioral life history. [Shows Jamie the following sequence.]

Q: How to test (sequence) ?

#1: enable->test->read->disable

#2: enable->test*[read]^n->test->disable

#3: [read]^n

#4: enable->disable->[test | read]

24.5.1 OOT Methods: Random Testing

- Random testing
  - identify operations applicable to a class
  - define constraints on their use
  - identify a minimum test sequence
    - an operation sequence that defines the minimum life history of the class (object)
  - generate a variety of random (but valid) test sequences
    - exercise other (more complex) class instance life histories

24.5.1 OOT Methods: Random Testing

Considering the Account class:

open(), setup(),
deposit() , withdraw()
balance(), summarize(), creditLimit(), close()

The minimum behavioral life history of an instance of Account:

open•setup•deposit•withdraw•close

open•setup•deposit•[deposit|withdraw|balance|summarize|creditLimit]^n•withdraw•close

Test case r_1: open•setup•deposit•deposit•balance•summarize•withdraw•close

Test case r_2: open•setup•deposit•withdraw•deposit•balance•creditLimit•withdraw•close

24.5.2 OOT Methods: Partition Testing

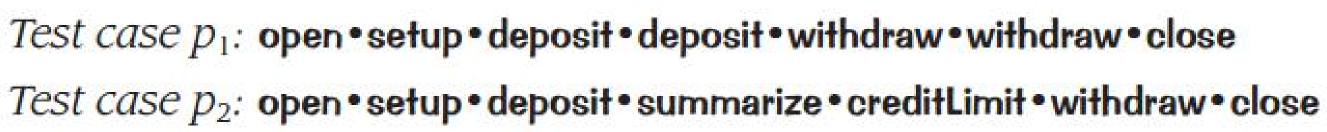
- Partition Testing
  - reduces the number of test cases required to test a class in much the same way as equivalence partitioning for conventional software
  - state-based partitioning
    - categorize and test operations based on their ability to change the state of a class
  - attribute-based partitioning
    - categorize and test operations based on the attributes that they use
  - category-based partitioning
    - categorize and test operations based on the generic function each performs

24.5.2 OOT Methods: Partition Testing

Considering the Account class

State-based partitioning

- state operations: deposit() and withdraw()
- nonstate operations:balance(), summarize(), and creditLimit()



Test case p_1: **open•setup•deposit•deposit•withdraw•withdraw•close**

Test case p_2: **open•setup•deposit•summarize•creditLimit•withdraw•close**

Attribute-based partitioning: creditLimit

- operations that use creditLimit
- operations that modify creditLimit
- operations that do not use or modify creditLimit

Category-based partitioning

- initialization operations(open, setup)
- computational operations (deposit,withdraw)
- queries (balance, summarize, creditLimit)
- termination operations (close)

24.6 OOT Methods: Inter-Class Testing

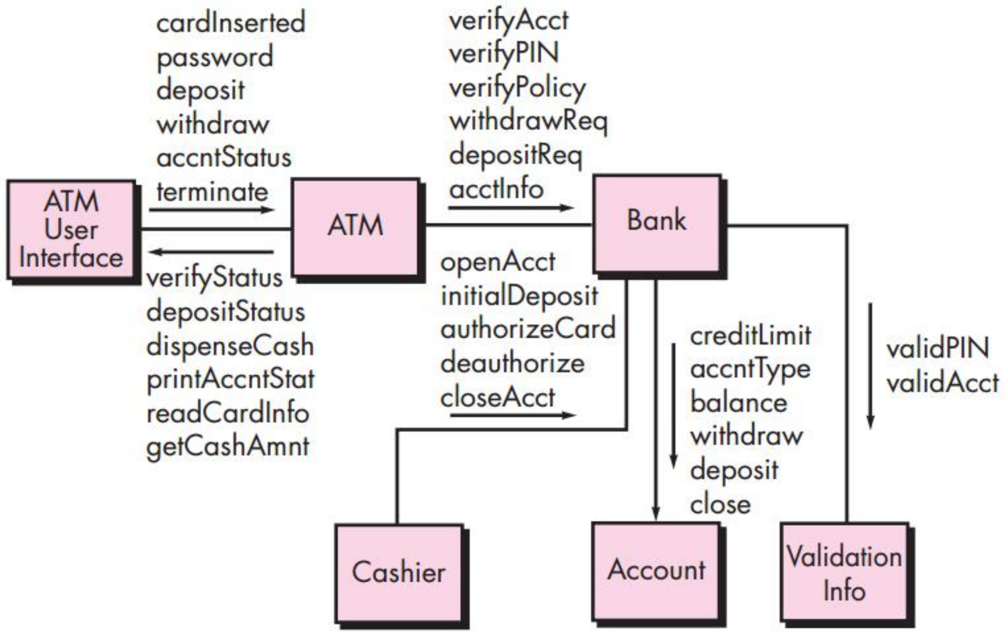
- Inter-class testing
  ① For each client class, use the list of class operators to generate a series of random test sequences. The operators will send messages to other server classes.
  ② For each message that is generated, determine the collaborator class and the corresponding operator in the server object.
  ③ For each operator in the server object (that has been invoked by messages sent from the client object), determine the messages that it transmits.
  ④ For each of the messages, determine the next level of operations that are invoked and incorporate these into the test sequence.

24.6 OOT Methods: Inter-Class Testing

consider a sequence of operations for the Bank class relative to an ATM class

verifyAcct•verifyPIN•[[verifyPolicy•withdrawReq]|depositReq|acctInfoREQ]^n

1. random: *Test case* r_3 = verifyAcct•verifyPIN•depositReq



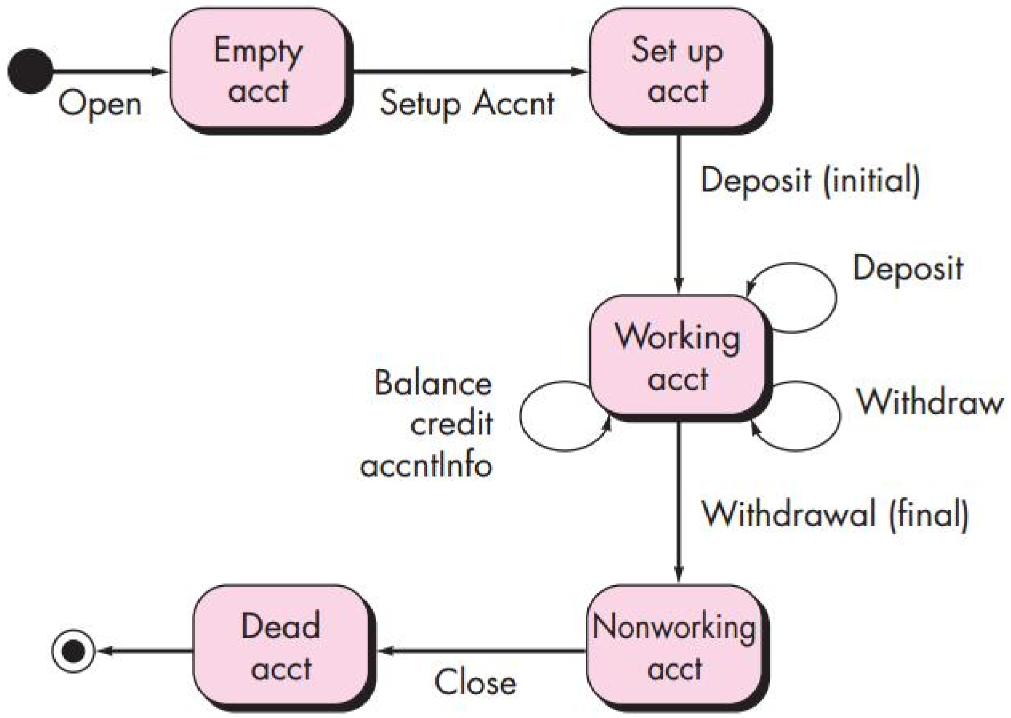
2. Collaborators:

- **Bank** must collaborate with **ValidationInfo** to execute the verifyAcct() and verifyPIN().
- **Bank** must collaborate with **Account** to execute depositReq().

Test case r_4 = verifyAcct [Bank:validAcctValidationInfo]•verifyPIN [Bank: validPinValidationInfo]•depositReq [Bank: depositaccount]

24.6 OOT Methods: Behavior Testing

The tests to be designed should achieve all state coverage. That is, the operation sequences should cause the Account class to make transition through all allowable states



Test case s₁: open•setupAccnt•deposit (initial)•withdraw (final)•close

minimum sequence

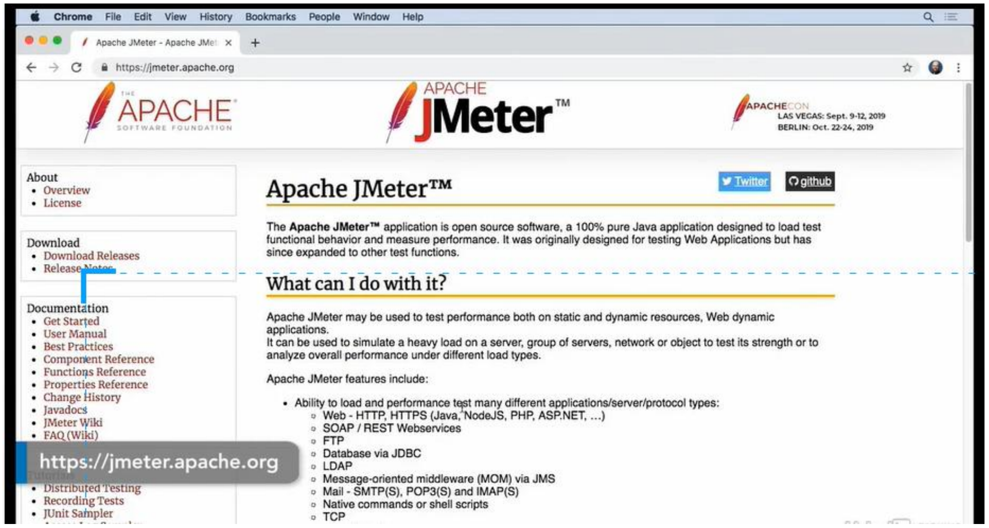
Test case s₂: open•setupAccnt•deposit(initial)•deposit•balance•credit•withdraw (final)•close

Test case s₃: open•setupAccnt•deposit(initial)•deposit•withdraw•accntInfo•withdraw (final)•close

System Testing

- Performance Testing
- Load Testing
- Interface Testing
- …..

Let's watch! (tool: JMeter)


Chrome File Edit View History Bookmarks People Window Help
Apache JMeter - Apache JMet
https://jmeter.apache.org
THE APACHE SOFTWARE FOUNDATION
APACHE JMeter™
APACHECON LAS VEGAS: Sept. 9-12, 2019 BERLIN: Oct. 22-24, 2019
Twitter
github
https://jmeter.apache.org

About
- Overview
- License

Download
- Download Releases
- Release Notes

Documentation
- Get Started
- User Manual
- Best Practices
- Component Reference
- Functions Reference
- Properties Reference
- Change History
- Javadocs
- JMeter Wiki
- FAQ (Wiki)

Tutorials
- Distributed Testing
- Recording Tests
- JUnit Sampler

Apache JMeter™

The **Apache JMeter™** application is open source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

What can I do with it?

Apache JMeter may be used to test performance both on static and dynamic resources, Web dynamic applications.
It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types.

Apache JMeter features include:

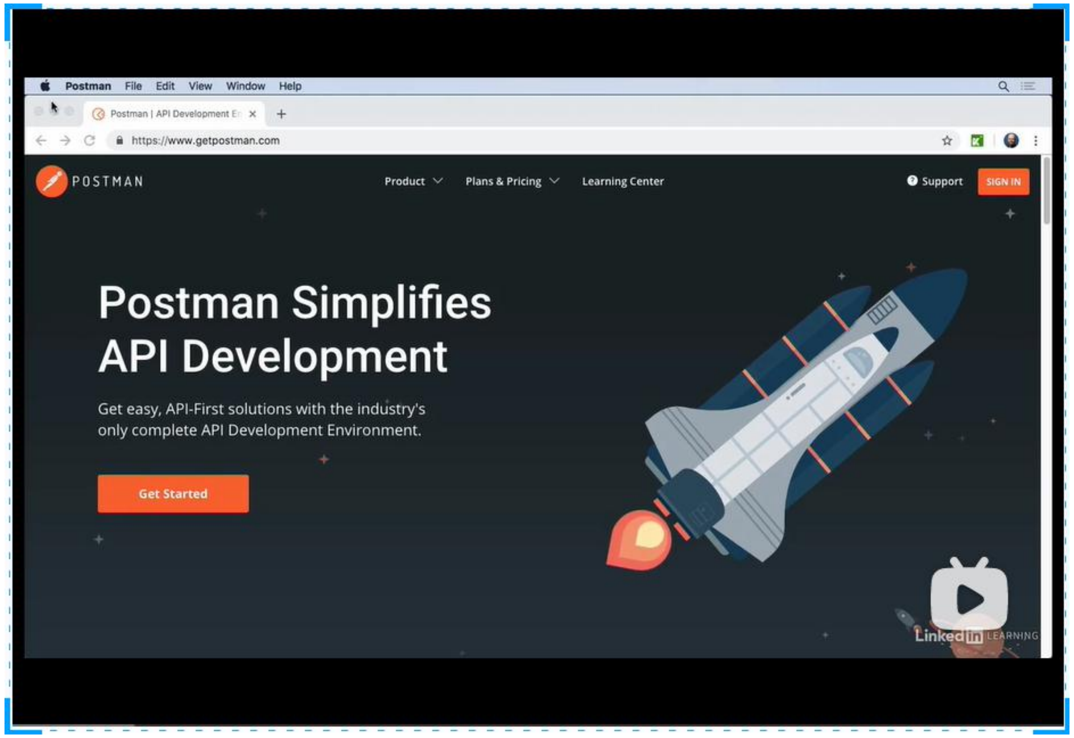
- Ability to load and performance test many different applications/server/protocol types:
  - Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, ...)
  - SOAP / REST Webservices
  - FTP
  - Database via JDBC
  - LDAP
  - Message-oriented middleware (MOM) via JMS
  - Mail - SMTP(S), POP3(S) and IMAP(S)
  - Native commands or shell scripts
  - TCP

https://www.bilibili.com/video/BV1Mz4y197ix?p=19&share_medium=android&share_plat=android&share_session_id=355d34c7-e33f-4e2a-b0d8-4c4056968d42&share_source=WEIXIN&share_tag=s_i&timestamp=1648601347&unique_k=4DbX2Si

System Testing

- Interface Testing

Let's watch! (tool: Postman)


Postman File Edit View Window Help
Postman | API Development E
https://www.getpostman.com
POSTMAN
Product
Plans & Pricing
Learning Center
Support
SIGN IN
Postman Simplifies API Development
Get easy, API-First solutions with the industry's only complete API Development Environment.
Get Started
Linked in LEARNING

https://www.bilibili.com/video/BV1Mz4y197ix?p=14

Summary

OO testing

1. Unit testing: Intra-class testing
2. Integration testing[inter-class testing]: thread-based testing, use-based testing,cluster testing
3. Validation testing: use-case in requirement, black-box testing
4. Methods:
   - Partition: State-based partitioning, Attribute-base, Category-based
   - Inheritance: superclass and subclass
     - ✓ if change some method m() in a superclass, we need to retest m() inside all subclass inherit it
     - ✓ if we change a subclass, we need to retest all related methods inherited from its superclass
   - Sequence (Random testing)
   - Behavior (state change)

THE END