投票 最多可选1项

设置

Are you ready?

A Yes

B No

提交

Review： ch19 software quality

- Definition: effective software process, useful product, measurable value
- Quality Model: McCall’s Quality Factors,
- ISO 9126 Quality model:Functionality, Reliability, Usability, Efficiency ,Maintainability , Portability
- Cost of Quality: Prevention costs , Appraisal costs , Internal failure costs, External failure costs
- Quality Control:
  - ✓ method: checklist, Pareto principle, histogram, fishbone graph, etc.

Review：ch21 SQA

- **Methods of SQA**
  - ✓ Defect prevention: Error blocking, Error source removal, process improvement
  - ✓ Defect removal: inspection and review, testing
  - ✓ Defect tolerance: NVP, out-voted,
  - ✓ recovery: recovery from failure
- **SQA Group**
  - ✓ roles: process review, work products review, audit, record and report.
  - ✓ goals: Quality of requirements, design, code; Quality control effectiveness
  - ✓ methods: review, statistical analysis
  - ✓ Six Sigma
- Software Reliability: MTTF, R = MTTF/(1+MTTF)

Software Engineering

Part 3
Quality Management

Chapter 22
Software Testing Strategies

Contents

Contents

单选题 1分

设置

此题未设置答案，请点击右侧设置按钮

What is the objective of testing?

A discover faults

B prove correctness

提交

22.1 Software Testing

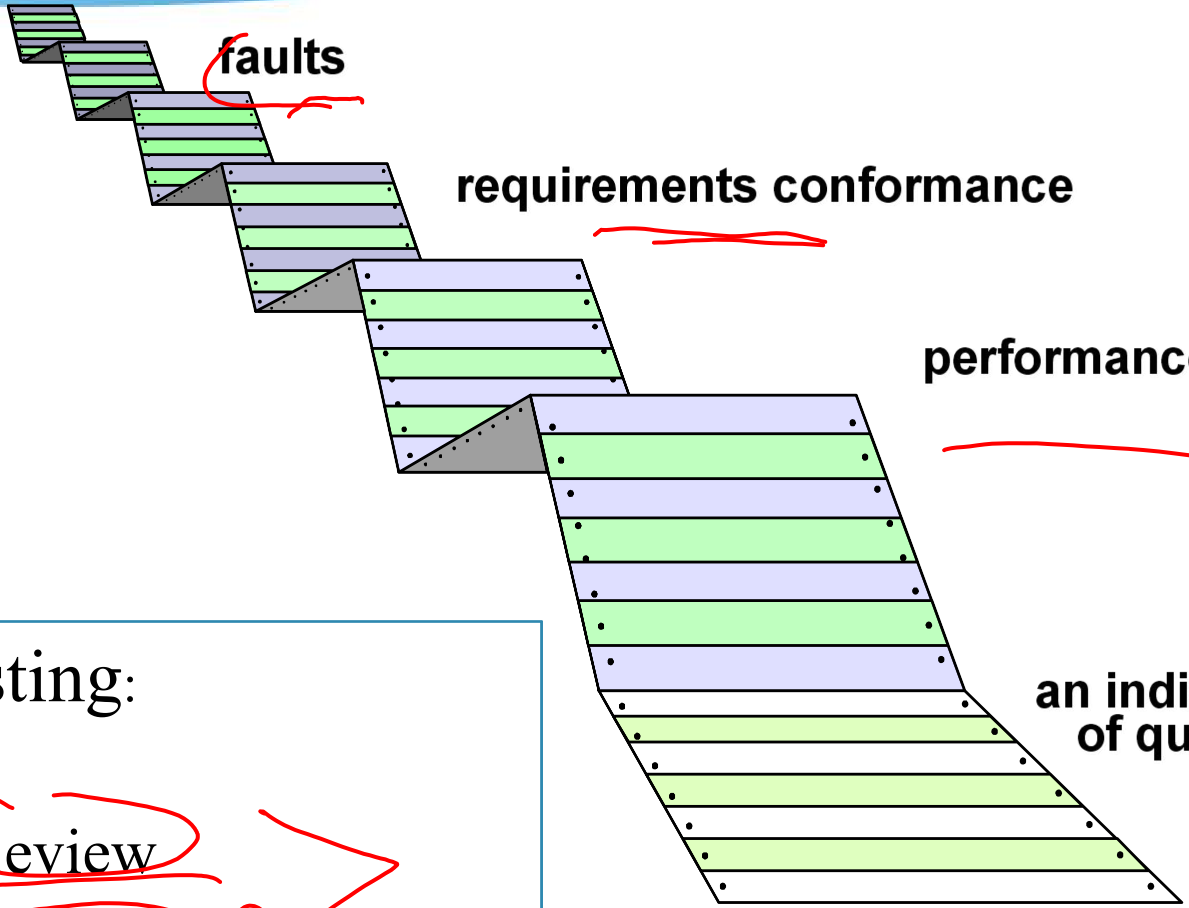
3 > 2

- Objective of testing:

    discover faults (> prove correctness)

- A test is successful only when a fault is discovered
    - **Fault identification** is the process of determining what fault caused the failure
    - **Fault correction** is the process of making changes to the system so that the faults are removed

22.1 What Testing Shows



Generalized testing:

- ✓ Static: Review
- ✓ Dynamic : Testing
- ✓ Correctness proof (formal)

22.1 Strategic Approach

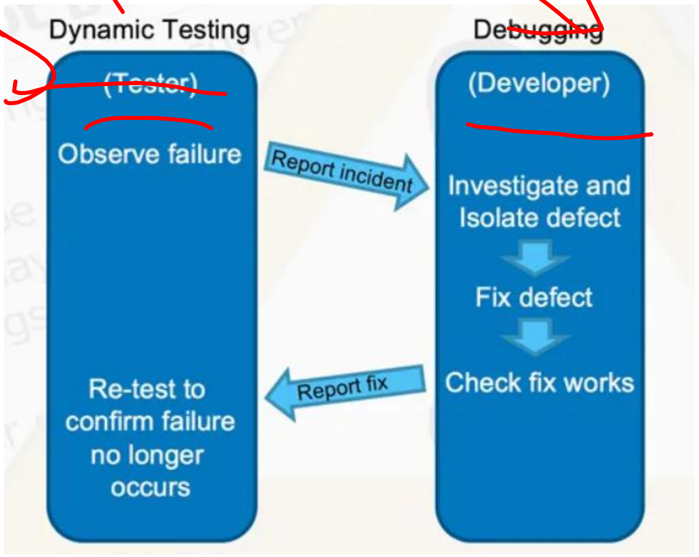
- To perform effective testing, you should conduct effective technical **reviews**. By doing this, many errors will be eliminated before testing commences.
- Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
- Different testing techniques 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.

https://medium.com/fintegro-company-inc/who-does-the-software-testing-developer-vs-tester-infographics-1f9660091832

主观题 10分

设置

What is the difference between testing and debugging?



https://dzone.com/articles/the-differences-between-testing-and-debugging

正常使用主观题需2.0以上版本雨课堂

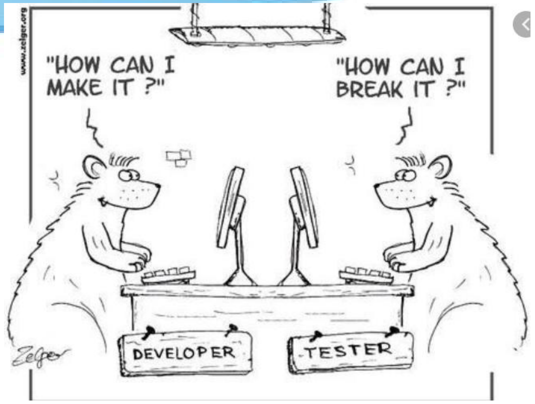
作答

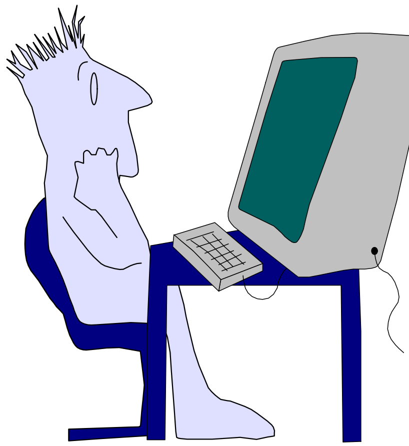
22.1.1 V & V

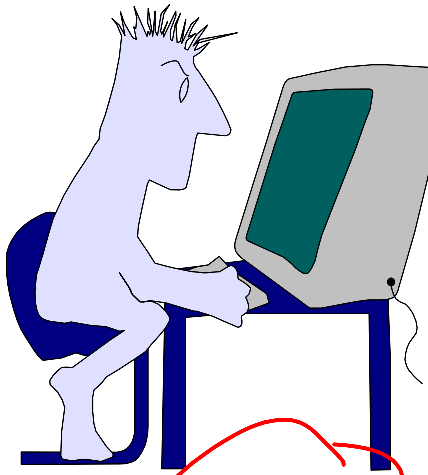
- *Validation:* Are we building the right product? (object correct)
- *Verification:* Are we building the product right? (technique correct)

- *Validation* refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.
- *Verification* refers to the set of tasks that ensure that software correctly implements a specific function.

22.1.2 Who Tests the Software?







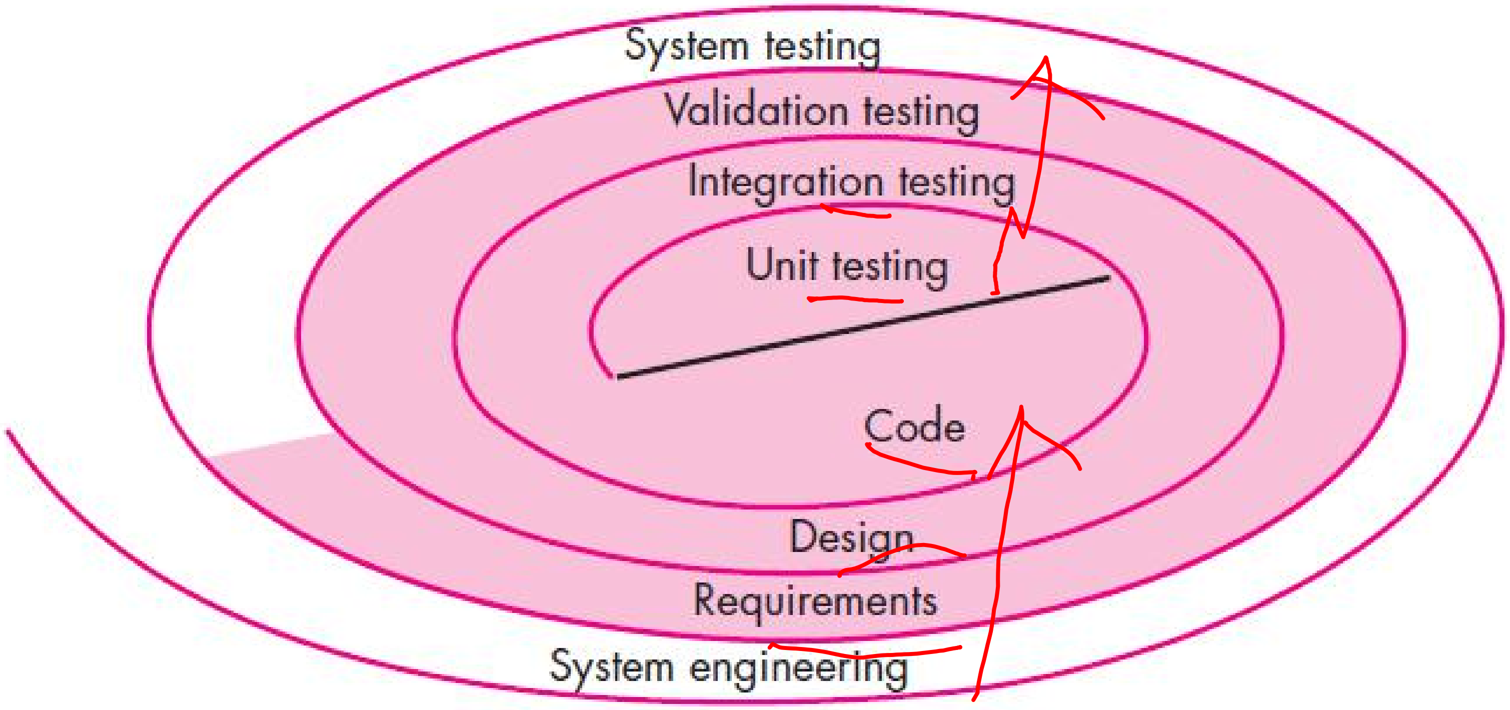
developer

independent tester

Understands the system
but, will test "gently"
and, is driven by "delivery"

Must learn about the system,
but, will attempt to break it
and, is driven by quality

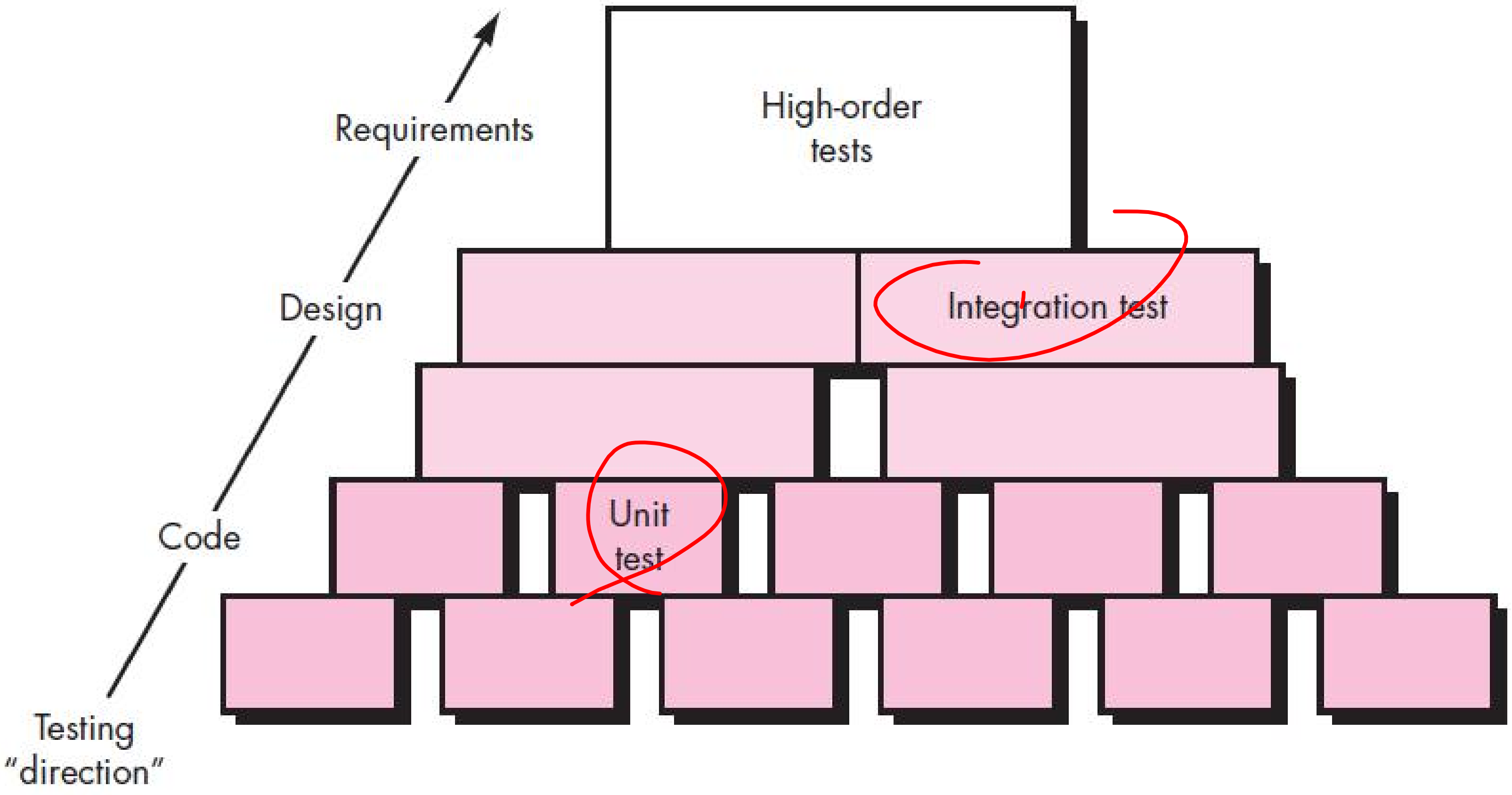
22.1.3 Testing Strategy - Big Picture



22.1.3 Testing Strategy

- We begin by ‘testing-in-the-small’ and move toward ‘testing-in-the-large’
- For conventional software
  - The **module** (component) is our initial focus
  - Integration of modules follows
- For OO software
  - our focus when “testing in the small” changes from an **individual** module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

22.1.3 Testing Steps

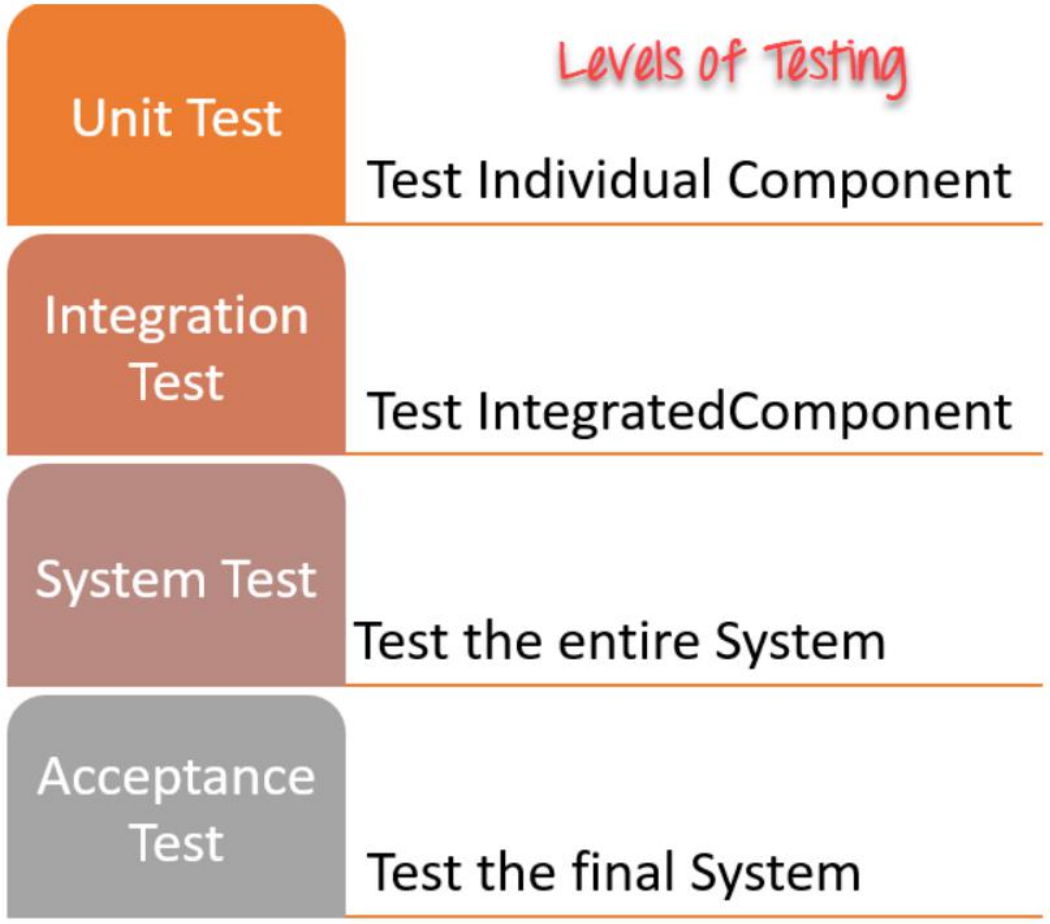


22.1.3 Strategic Issues

Effectiveness of fault-discovery techniques

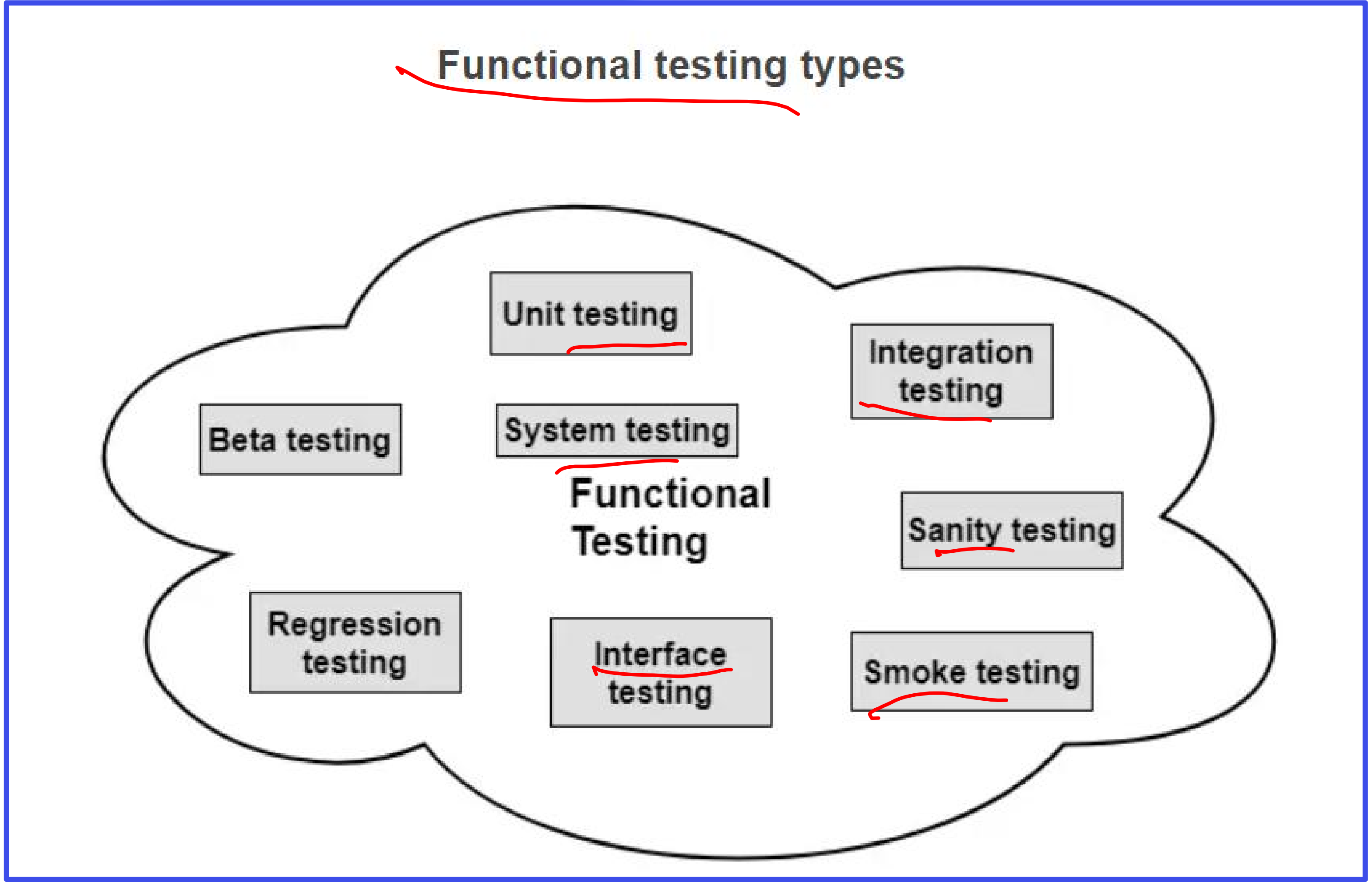
	Requirements Faults	Design Faults	Code Faults	Documentation Faults
Reviews	Fair	Excellent	Excellent	Good
Prototypes	Good	Fair	Fair	Not applicable
Testing	Poor	Poor	Good	Fair
Correctness Proofs	Poor	Poor	Fair	Fair

22.1.4 Testing Type



https://www.guru99.com/levels-of-testing.html

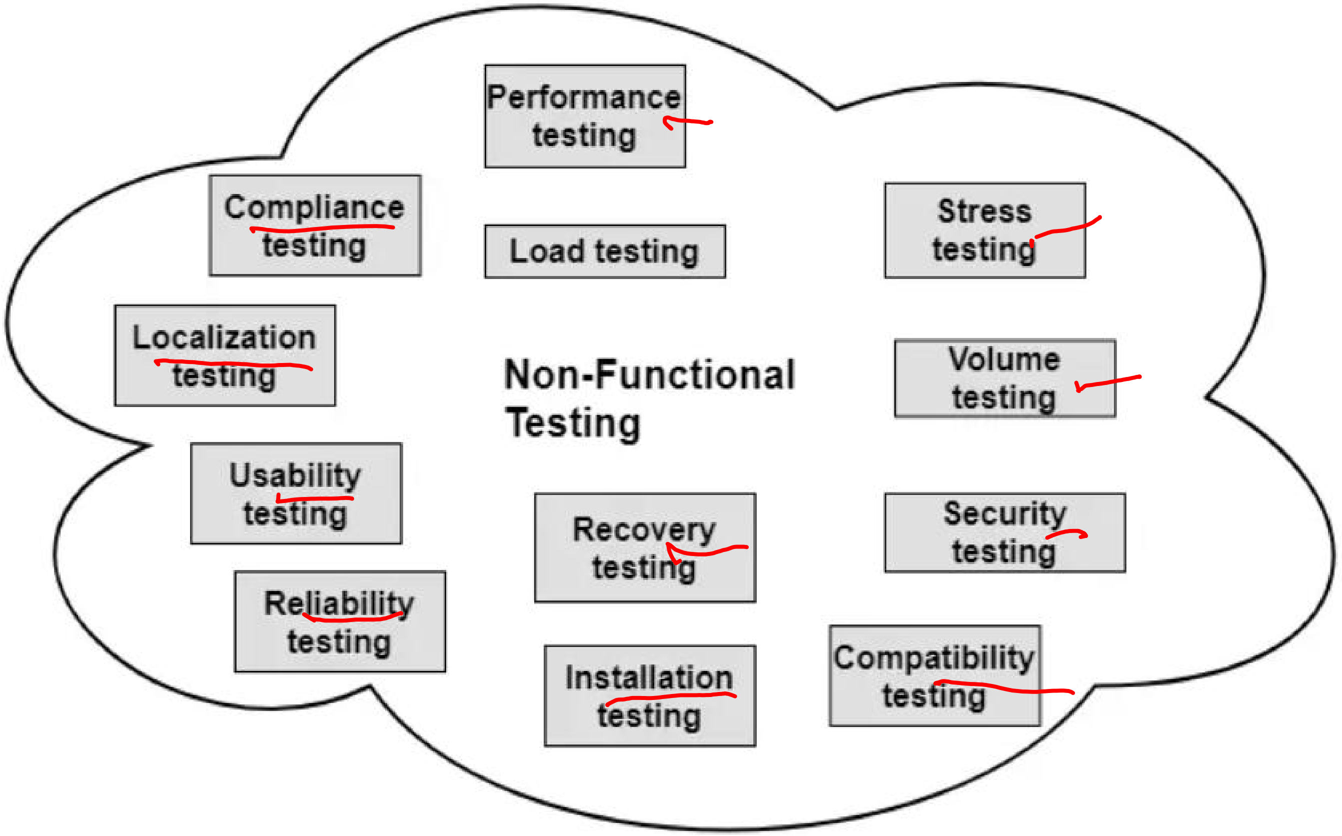
22.1.4 Testing Type



https://www.techbeamers.com/testing-types/

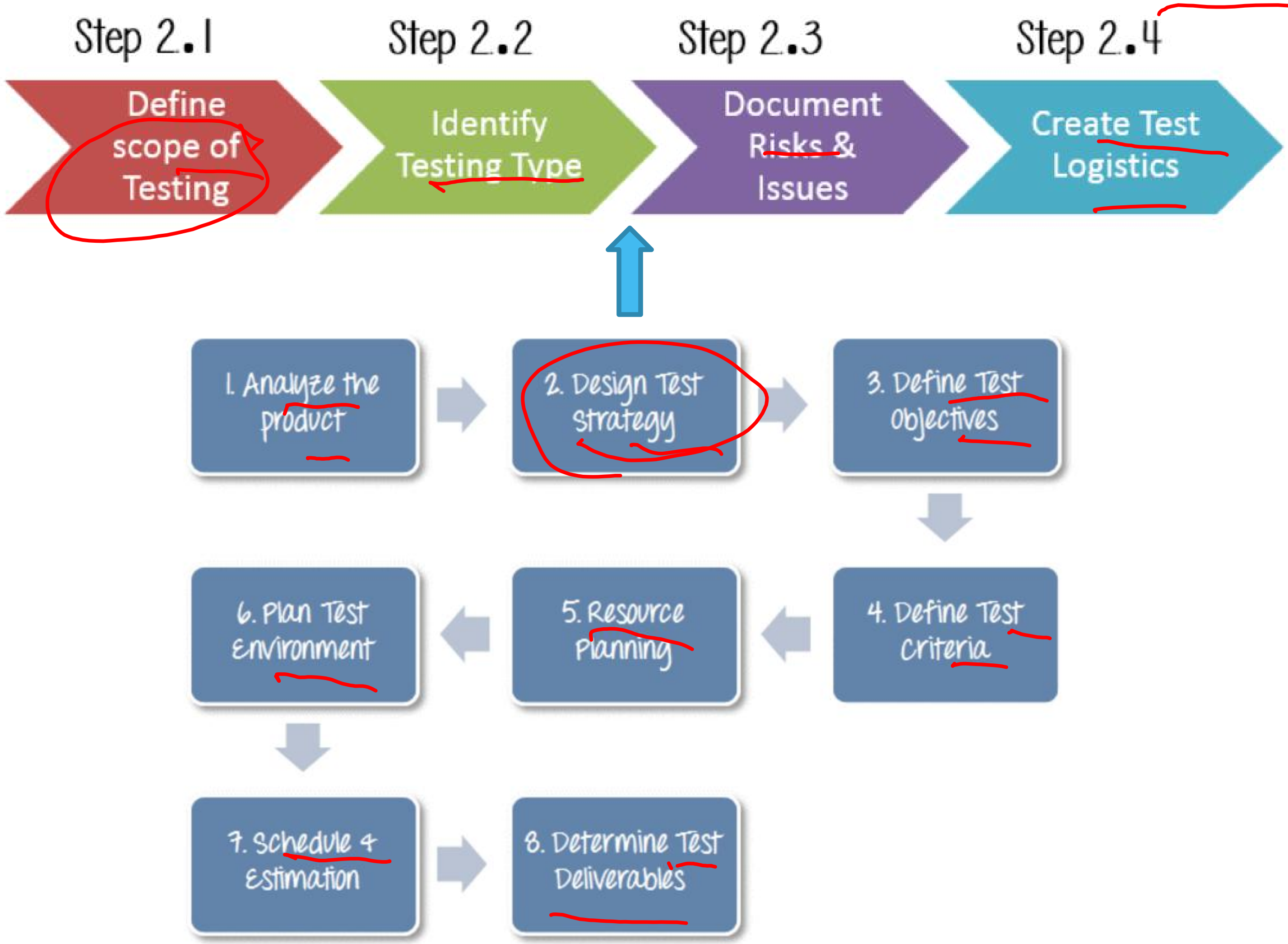
22.1.4 Testing Type

Non-functional testing types



22.1.5 Test Planning

QA

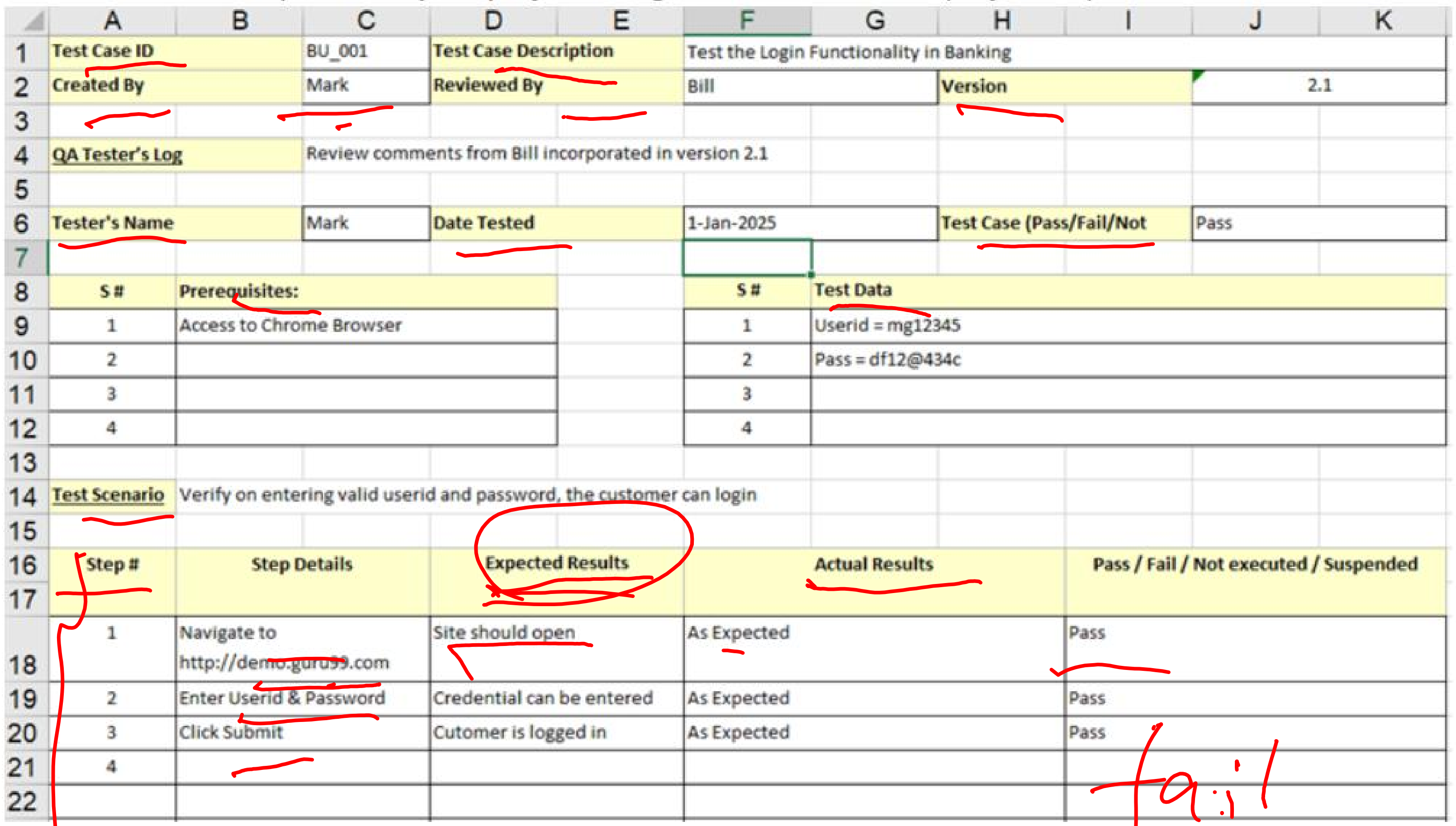


https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html

https://www.softwaretestinghelp.com/how-to-write-test-plan-document-software-testing-training-day3/

22.2 Test suite and Test case

Test case templet (example) [test suite is a set of test cases]

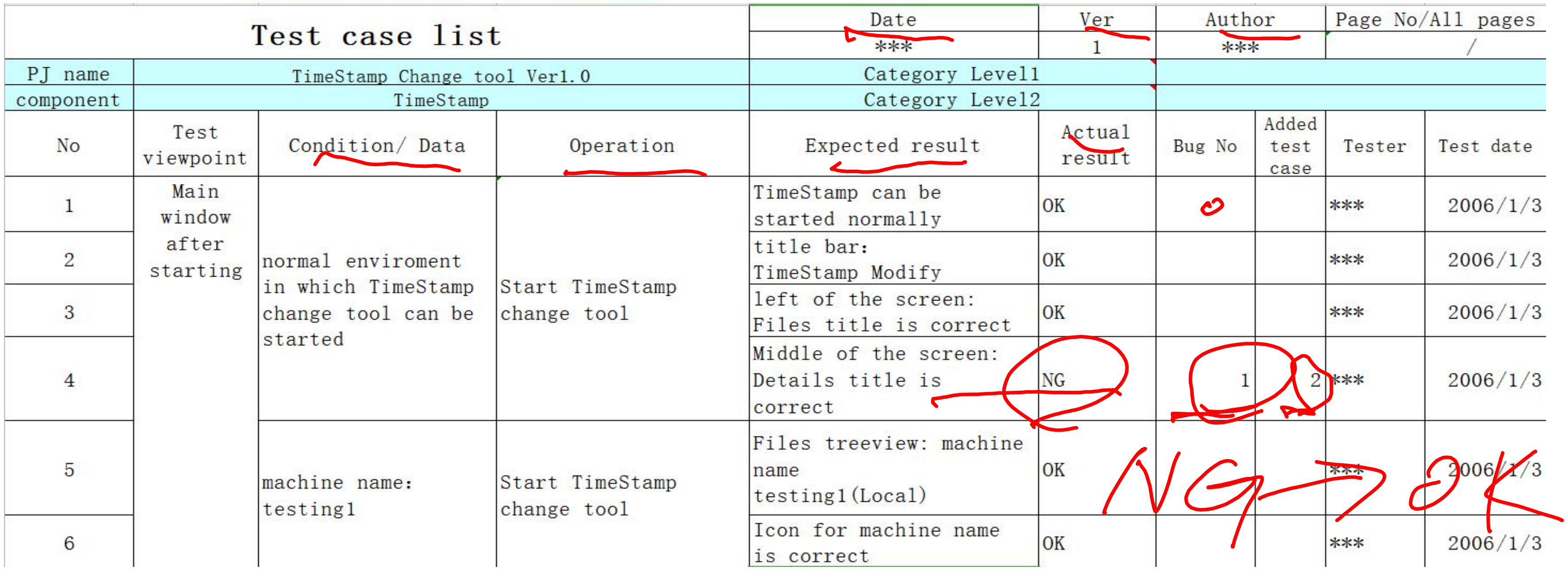


	A	B	C	D	E	F	G	H	I	J	K
1	Test Case ID		BU_001	Test Case Description		Test the Login Functionality in Banking					
2	Created By		Mark	Reviewed By		Bill		Version		2.1	
3											
4	QA Tester's Log		Review comments from Bill incorporated in version 2.1								
5											
6	Tester's Name		Mark	Date Tested		1-Jan-2025		Test Case (Pass/Fail/Not		Pass	
7											
8	S #	Prerequisites:				S #	Test Data				
9	1	Access to Chrome Browser				1	Userid = mg12345				
10	2					2	Pass = df12@434c				
11	3					3					
12	4					4					
13											
14	Test Scenario	Verify on entering valid userid and password, the customer can login									
15											
16-17	Step #	Step Details		Expected Results		Actual Results			Pass / Fail / Not executed / Suspended		
18	1	Navigate to http://demo.guru99.com		Site should open		As Expected			Pass		
19	2	Enter Userid & Password		Credential can be entered		As Expected			Pass		
20	3	Click Submit		Cutomer is logged in		As Expected			Pass		
21	4										
22											

fail

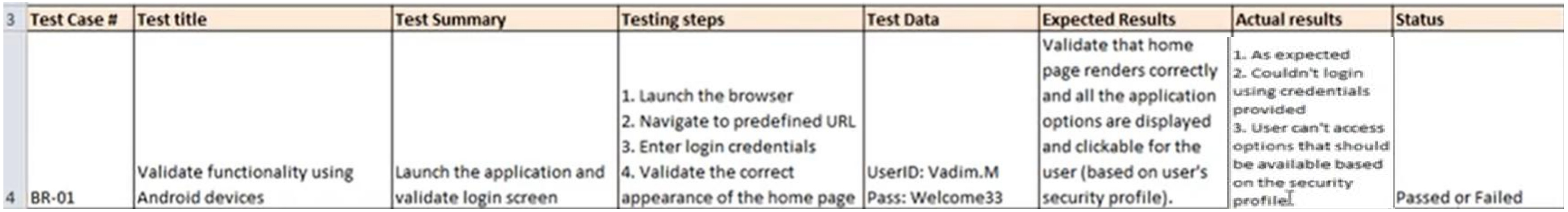
https://www.guru99.com/download-sample-test-case-template-with-explanation-of-important-fields.html

22.2 Test suite and Test case



Test case list				Date	Ver	Author		Page No/All pages	
				***	1	***		/	
PJ name	TimeStamp Change tool Ver1.0			Category Level1					
component	TimeStamp			Category Level2					
No	Test viewpoint	Condition/ Data	Operation	Expected result	Actual result	Bug No	Added test case	Tester	Test date
1	Main window after starting	normal enviroment in which TimeStamp change tool can be started	Start TimeStamp change tool	TimeStamp can be started normally	OK			***	2006/1/3
2				title bar: TimeStamp Modify	OK			***	2006/1/3
3				left of the screen: Files title is correct	OK			***	2006/1/3
4				Middle of the screen: Details title is correct	NG	1	2	***	2006/1/3
5		machine name: testing1	Start TimeStamp change tool	Files treeview: machine name testing1(Local)	OK			***	2006/1/3
6				Icon for machine name is correct	OK			***	2006/1/3

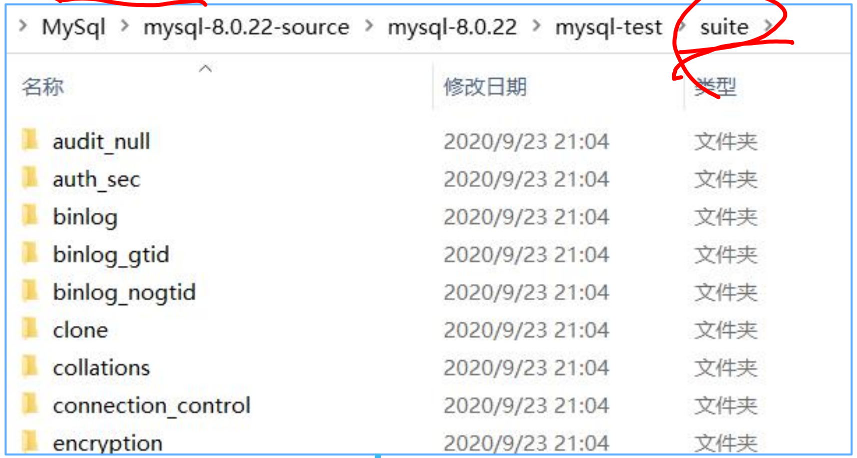
NG → OK



3	Test Case #	Test title	Test Summary	Testing steps	Test Data	Expected Results	Actual results	Status
4	BR-01	Validate functionality using Android devices	Launch the application and validate login screen	1. Launch the browser 2. Navigate to predefined URL 3. Enter login credentials 4. Validate the correct appearance of the home page	UserID: Vadim.M Pass: Welcome33	Validate that home page renders correctly and all the application options are displayed and clickable for the user (based on user's security profile).	1. As expected 2. Couldn't login using credentials provided 3. User can't access options that should be available based on the security profile	Passed or Failed

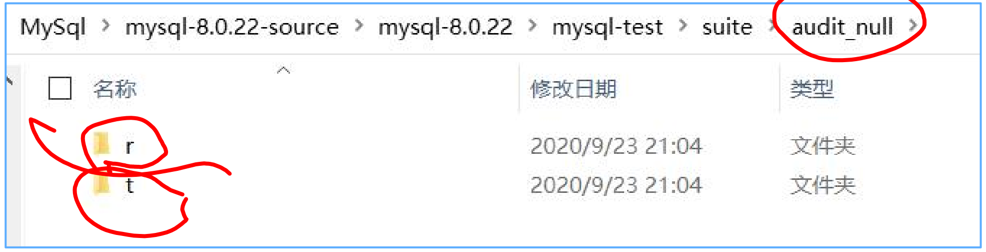
22.2 Test suite and Test case

MySQL：Test suite



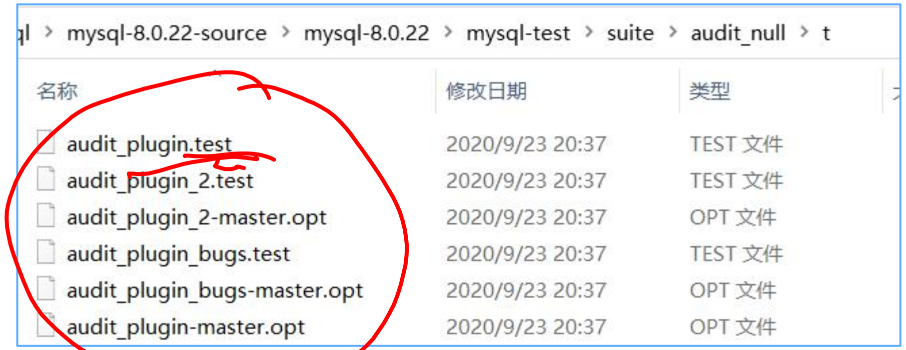
› MySql › mysql-8.0.22-source › mysql-8.0.22 › mysql-test › suite ›

名称	修改日期	类型
audit_null	2020/9/23 21:04	文件夹
auth_sec	2020/9/23 21:04	文件夹
binlog	2020/9/23 21:04	文件夹
binlog_gtid	2020/9/23 21:04	文件夹
binlog_nogtid	2020/9/23 21:04	文件夹
clone	2020/9/23 21:04	文件夹
collations	2020/9/23 21:04	文件夹
connection_control	2020/9/23 21:04	文件夹
encryption	2020/9/23 21:04	文件夹



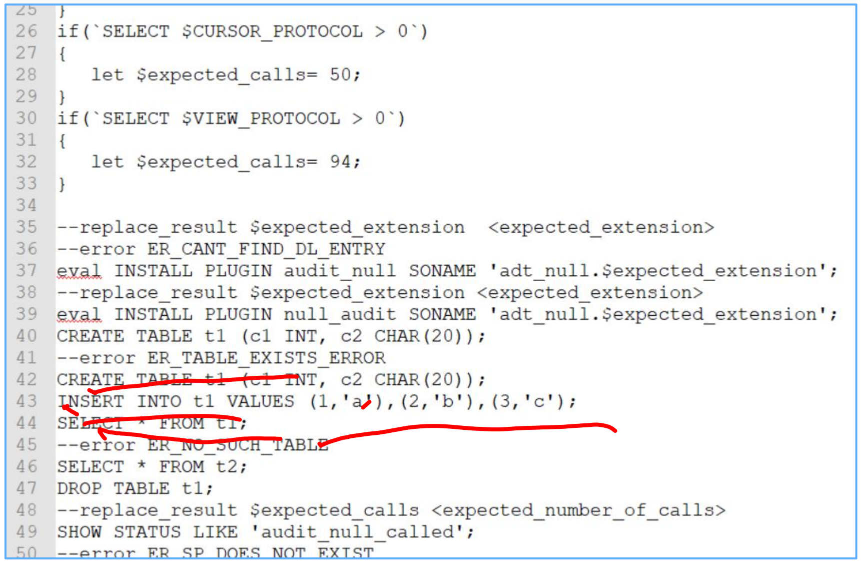
MySql › mysql-8.0.22-source › mysql-8.0.22 › mysql-test › suite › audit_null ›

名称	修改日期	类型
r	2020/9/23 21:04	文件夹
t	2020/9/23 21:04	文件夹



ql › mysql-8.0.22-source › mysql-8.0.22 › mysql-test › suite › audit_null › t

名称	修改日期	类型
audit_plugin.test	2020/9/23 20:37	TEST 文件
audit_plugin_2.test	2020/9/23 20:37	TEST 文件
audit_plugin_2-master.opt	2020/9/23 20:37	OPT 文件
audit_plugin_bugs.test	2020/9/23 20:37	TEST 文件
audit_plugin_bugs-master.opt	2020/9/23 20:37	OPT 文件
audit_plugin-master.opt	2020/9/23 20:37	OPT 文件



```
}
if(`SELECT $CURSOR_PROTOCOL > 0`)
{
   let $expected_calls= 50;
}
if(`SELECT $VIEW_PROTOCOL > 0`)
{
   let $expected_calls= 94;
}

--replace_result $expected_extension  <expected_extension>
--error ER_CANT_FIND_DL_ENTRY
eval INSTALL PLUGIN audit_null SONAME 'adt_null.$expected_extension';
--replace_result $expected_extension <expected_extension>
eval INSTALL PLUGIN null_audit SONAME 'adt_null.$expected_extension';
CREATE TABLE t1 (c1 INT, c2 CHAR(20));
--error ER_TABLE_EXISTS_ERROR
CREATE TABLE t1 (c1 INT, c2 CHAR(20));
INSERT INTO t1 VALUES (1,'a'),(2,'b'),(3,'c');
SELECT * FROM t1;
--error ER_NO_SUCH_TABLE
SELECT * FROM t2;
DROP TABLE t1;
--replace_result $expected_calls <expected_number_of_calls>
SHOW STATUS LIKE 'audit_null_called';
--error ER_SP_DOES_NOT_EXIST
```

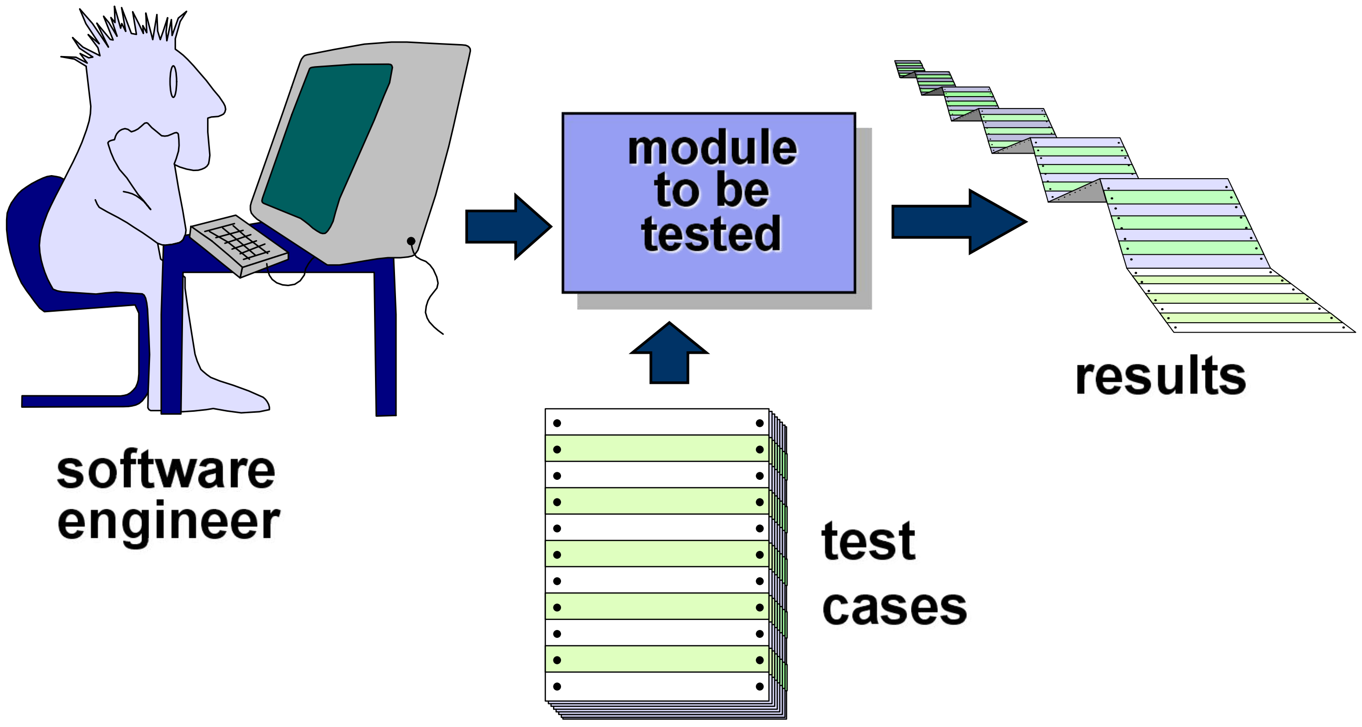
Test case

Take a break

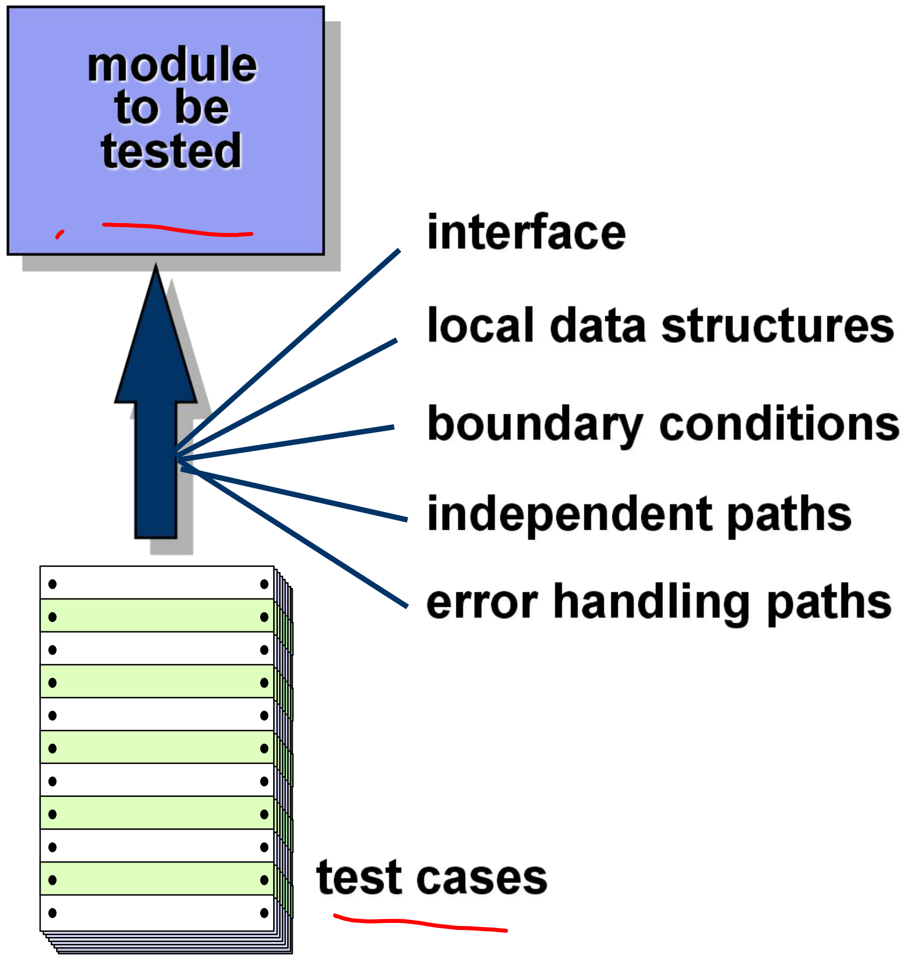
Five minutes

22.3.1 Unit Testing

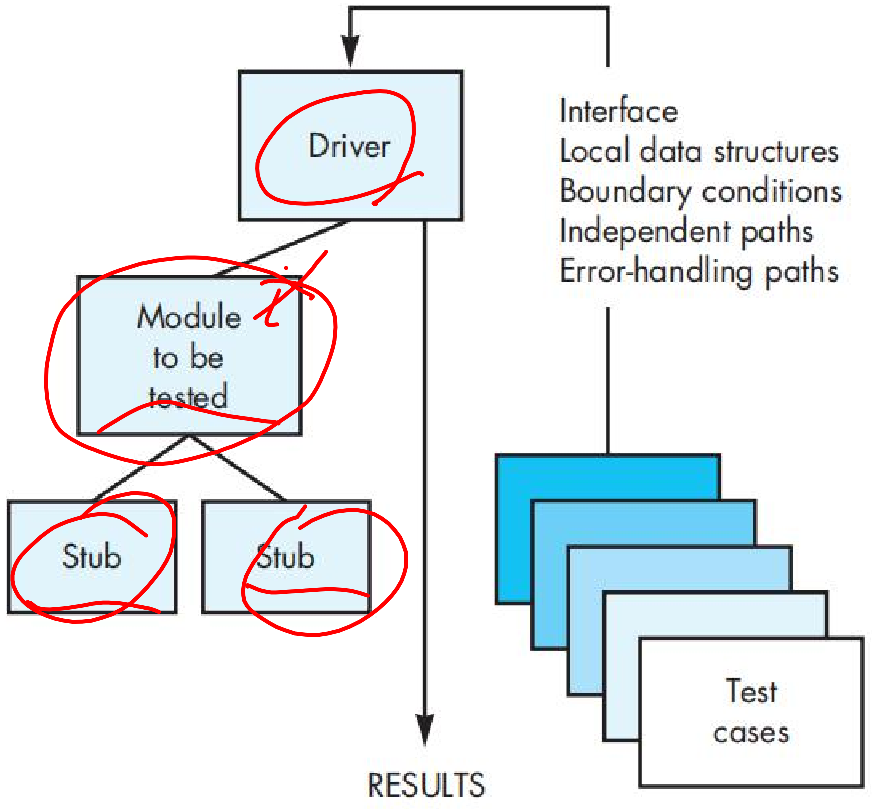
Unit testing focuses verification effort on the smallest unit of software design—the software component or module



22.3.1 Unit Testing



22.3.1 Unit Test Environment



test cases

```
void displayInfo{
....
getStudAddress(ID: Integer)
....
}
```

22.3.1 Unit Test Example

Python

```python
import unittest


class TestSum(unittest.TestCase):

    def test_sum(self):
        self.assertEqual(sum([1, 2, 3]), 6, "Should be 6")

    def test_sum_tuple(self):
        self.assertEqual(sum((1, 2, 2)), 6, "Should be 6")

if __name__ == '__main__':
    unittest.main()
```

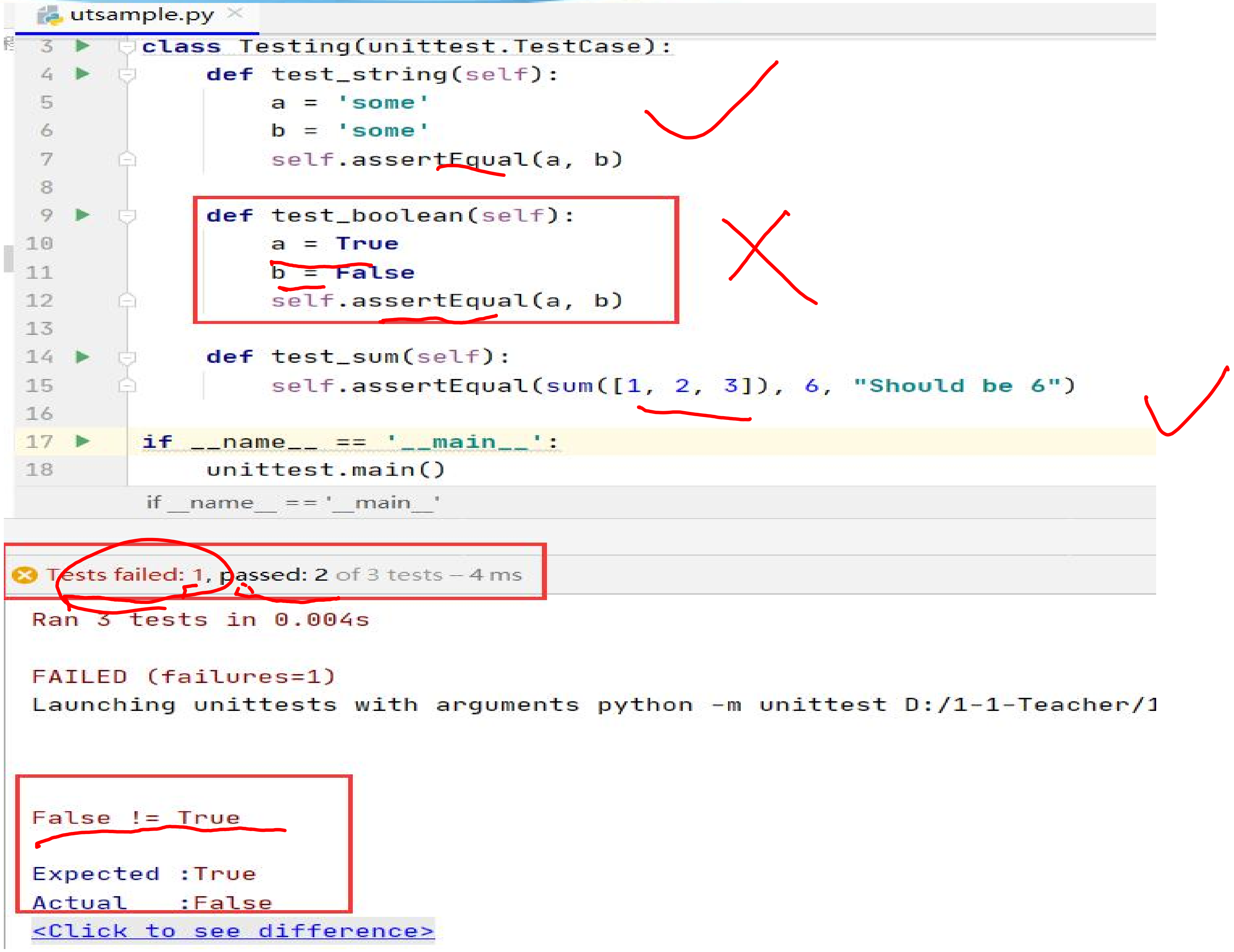
pass

fail

22.3.1 Unit Test Example

Method	Equivalent to
`.assertEqual(a, b)`	`a == b`
`.assertTrue(x)`	`bool(x) is True`
`.assertFalse(x)`	`bool(x) is False`
`.assertIs(a, b)`	`a is b`
`.assertIsNone(x)`	`x is None`
`.assertIn(a, b)`	`a in b`
`.assertIsInstance(a, b)`	`isinstance(a, b)`

22.3.1 Unit Test Example



utsample.py

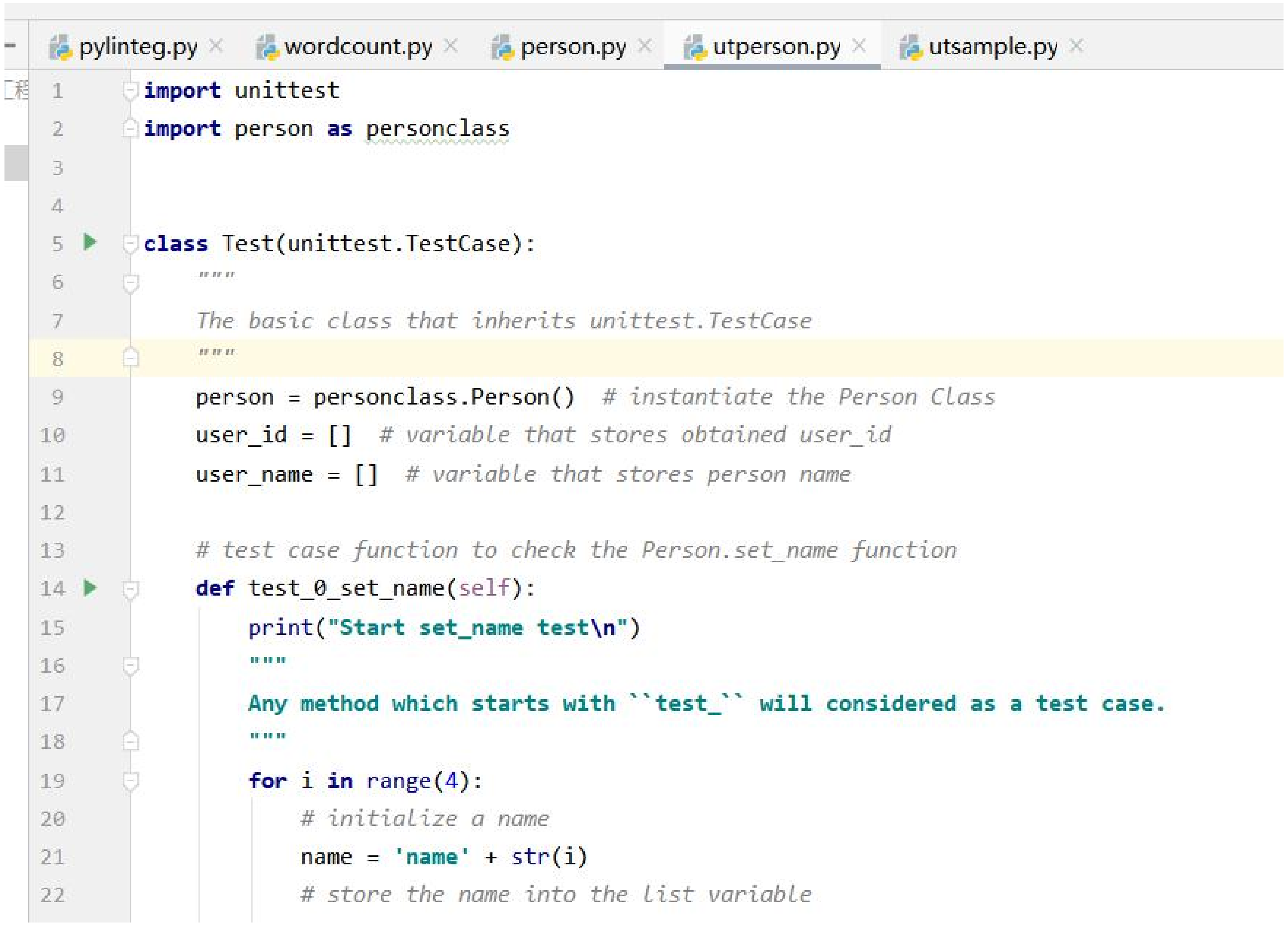
```
class Testing(unittest.TestCase):
    def test_string(self):
        a = 'some'
        b = 'some'
        self.assertEqual(a, b)

    def test_boolean(self):
        a = True
        b = False
        self.assertEqual(a, b)

    def test_sum(self):
        self.assertEqual(sum([1, 2, 3]), 6, "Should be 6")

if __name__ == '__main__':
    unittest.main()
```

if __name__ == '__main__'

```
Tests failed: 1, passed: 2 of 3 tests – 4 ms

Ran 3 tests in 0.004s

FAILED (failures=1)
Launching unittests with arguments python -m unittest D:/1-1-Teacher/1


False != True

Expected :True
Actual   :False
<Click to see difference>
```

22.3.1 Unit Test Example

PyCharm Demo



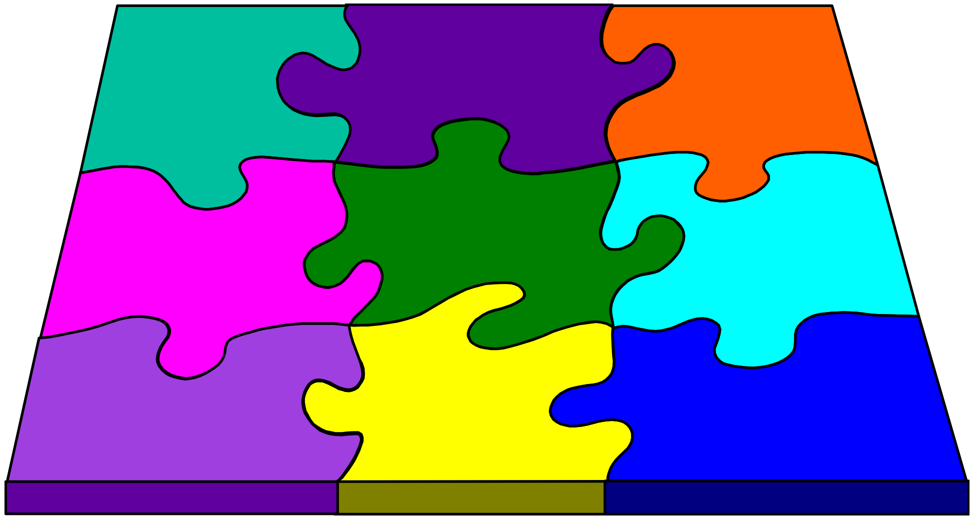
```python
import unittest
import person as personclass


class Test(unittest.TestCase):
    """
    The basic class that inherits unittest.TestCase
    """
    person = personclass.Person()  # instantiate the Person Class
    user_id = []  # variable that stores obtained user_id
    user_name = []  # variable that stores person name

    # test case function to check the Person.set_name function
    def test_0_set_name(self):
        print("Start set_name test\n")
        """
        Any method which starts with ``test_`` will considered as a test case.
        """
        for i in range(4):
            # initialize a name
            name = 'name' + str(i)
            # store the name into the list variable
```

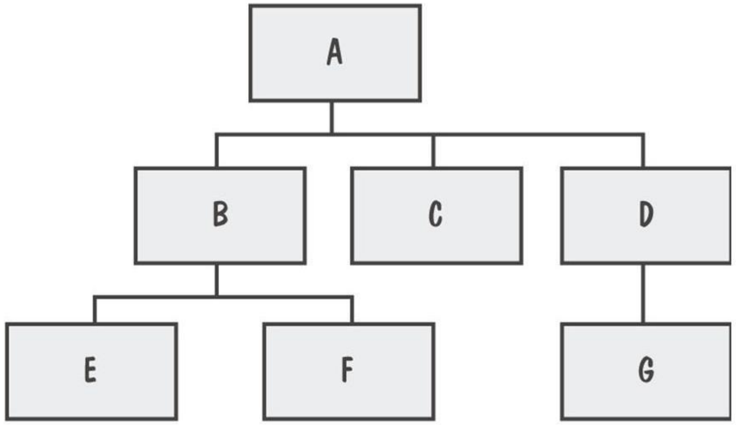
https://www.journaldev.com/15899/python-unittest-unit-test-example#python-unit-test-example-source
https://realpython.com/python-testing/

22.3.2 Integration Testing Strategies

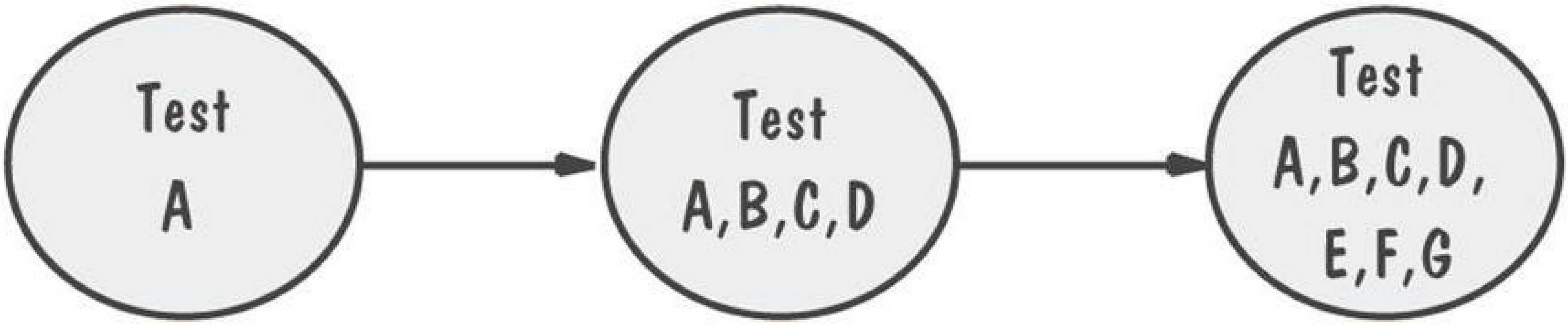
Options:

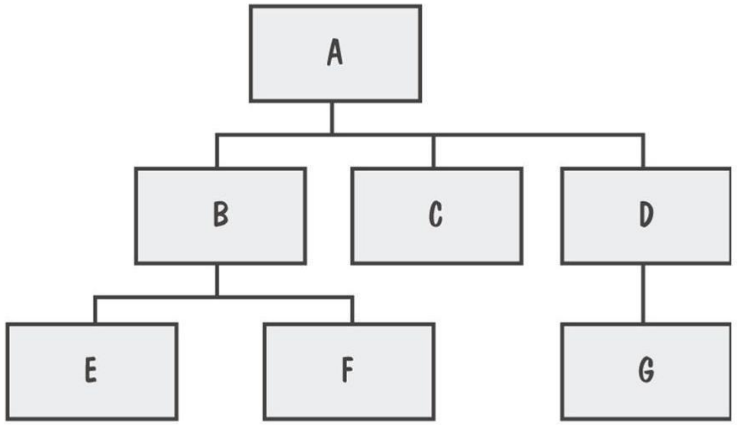
- the "big bang" approach
- an incremental construction strategy



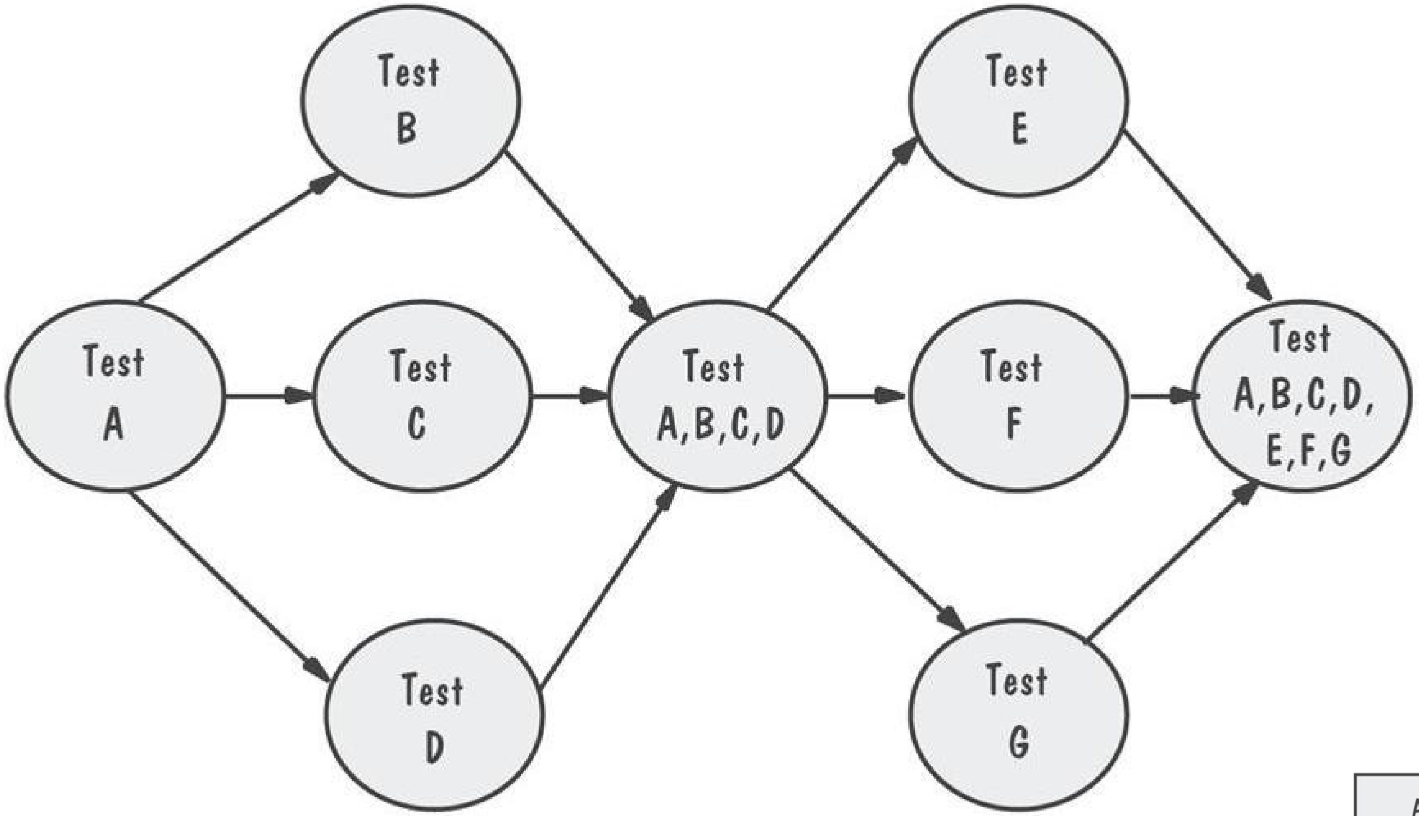


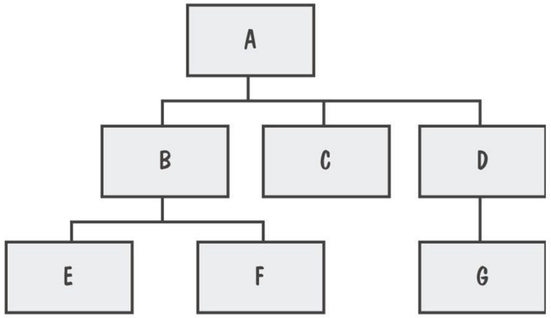
22.3.2 Top Down Integration





22.3.2 Modified Top-Down Integration

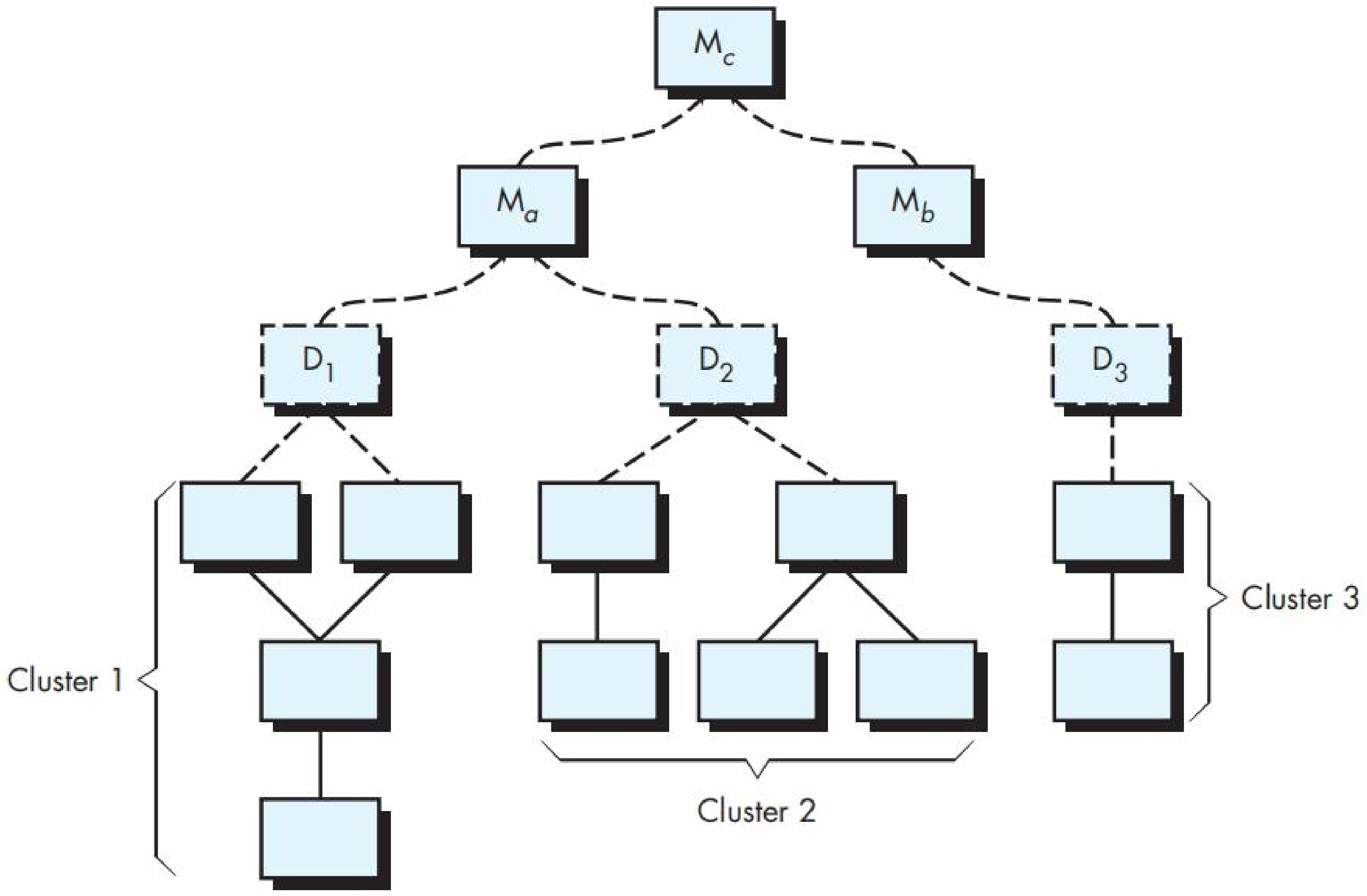




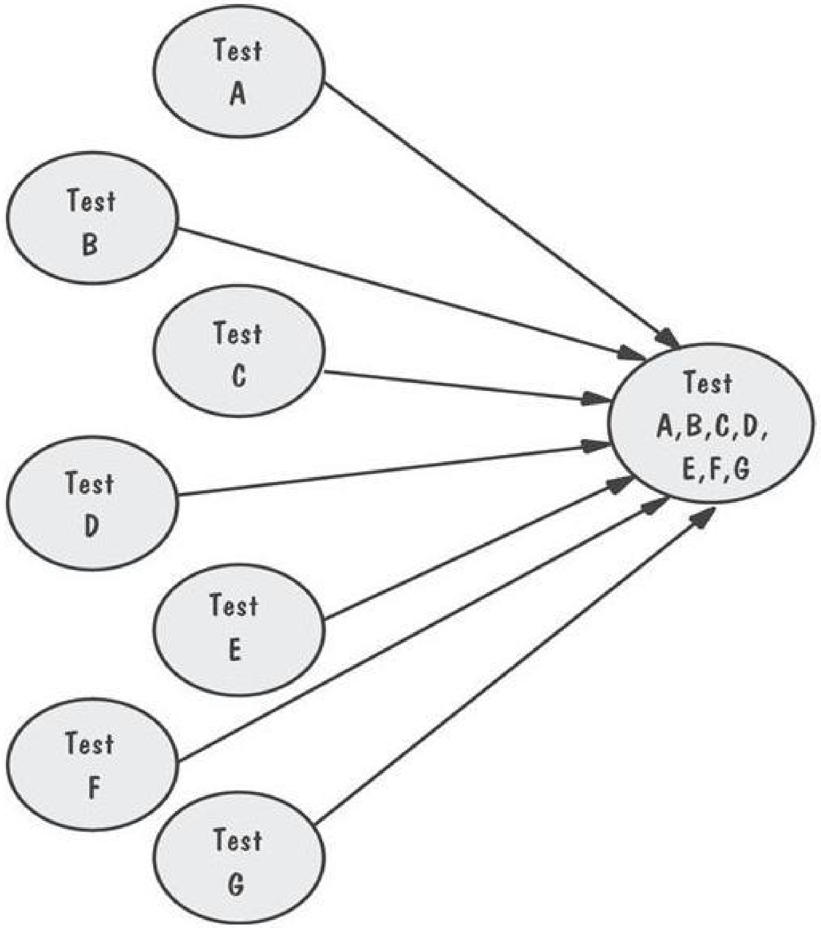
22.3.2 Bottom-Up Integration

FIGURE 22.6

Bottom-up integration

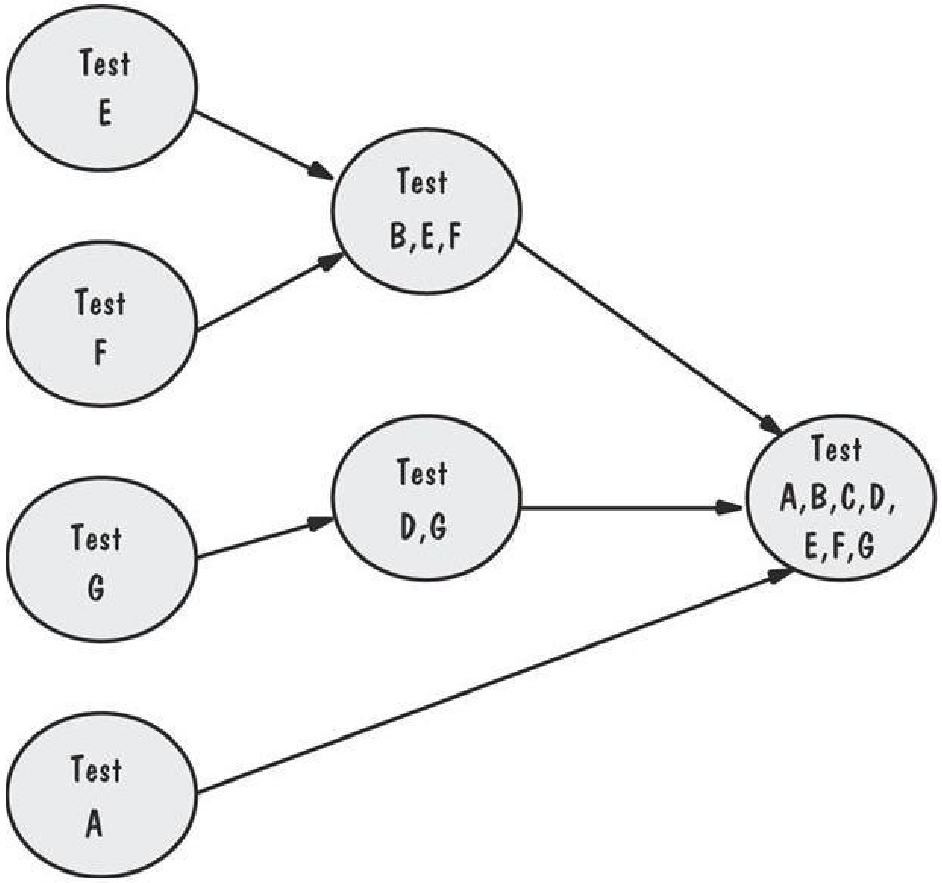


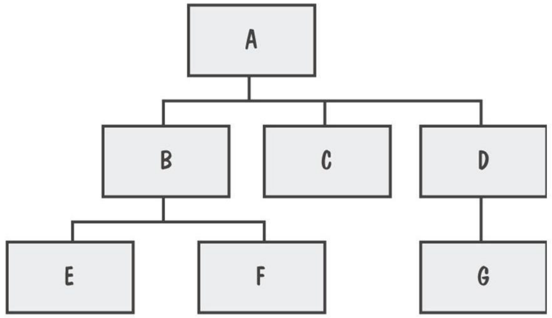
22.3.2 Bing-Bang Integration



22.3.2 Sandwich Integration

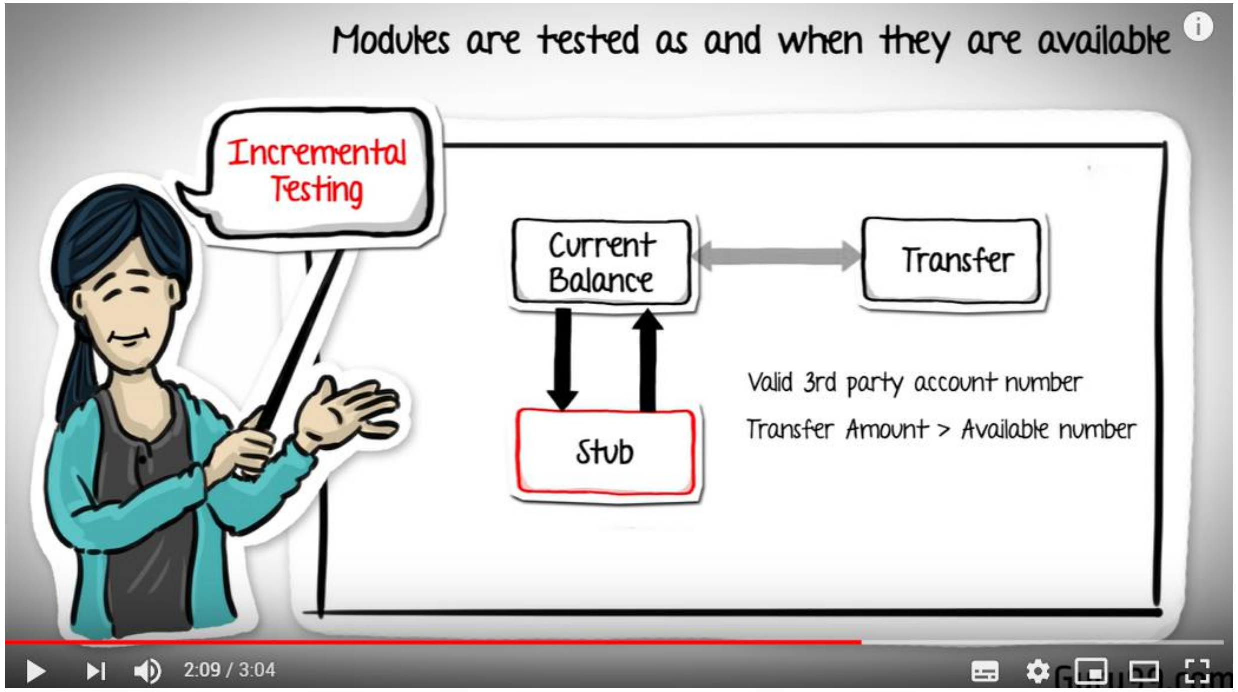
Viewed system as three layers





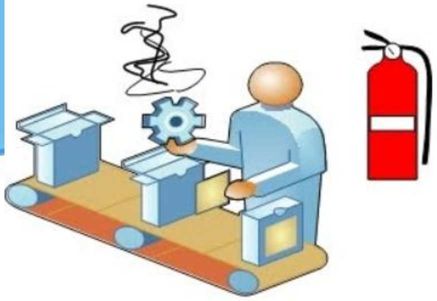
22.3.2 Integration Testing Sample

Stub and Driver - [Let' watch]


Modules are tested as and when they are available
Incremental Testing
Current Balance
Transfer
Stub
Valid 3rd party account number
Transfer Amount > Available number
2:09 / 3:04

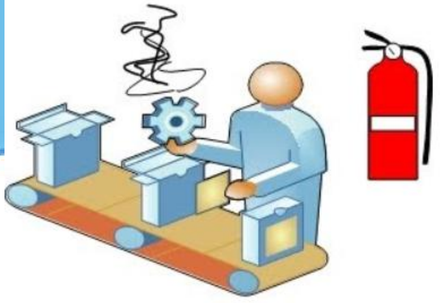
https://www.youtube.com/watch?v=QYCaaNz8emY

22.3.2 Smoke Testing



- A common approach for creating **"daily builds"** for product software (an integration testing approach).
- **Smoke testing steps**:
  - Software components are integrated into a "build."
    - A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
  - A series of tests is designed to expose errors that will keep the build from properly performing its function.
    - The intent should be to uncover **"show stopper" errors** that have the highest likelihood of throwing the software project behind schedule.

22.3.2 Sanity Testing



- **Sanity Testing:**
  - A test execution which is done to touch each implementation and its impact but not thoroughly or in-depth, it may include functional, UI, version, etc. testing depending on the implementation and its impact.

Smoke Testing Vs Sanity Testing: Difference

Tested by dev group (expose error)

Tested by QA or test group (check)

Take a break

Five minutes

22.4 Object-Oriented Testing

- Begins by evaluating the **correctness** and **consistency** of the analysis and design models
- testing strategy **changes**
  - An encapsulated class is usually the focus of unit testing.
  - integration focuses on classes and their execution across a ‘thread’ or in the context of a usage scenario
  - validation uses conventional black box methods

- test case design draws on conventional methods, but also encompasses special features

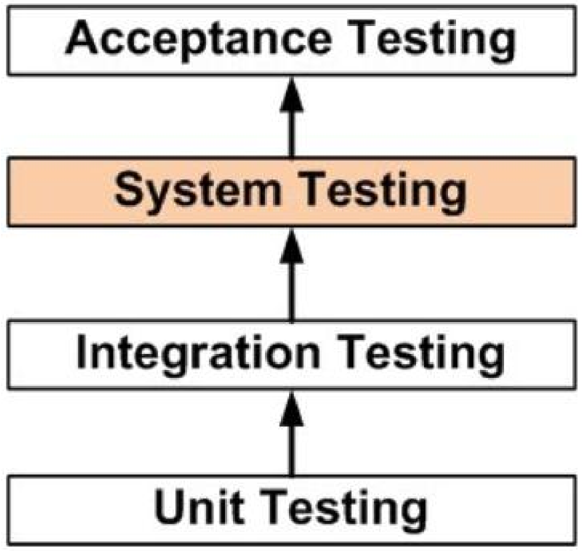
22.4 OO Testing Strategy

- **class testing is the equivalent of unit testing**
  - **operations** within the class are tested
  - **state** behavior of the class is examined

- **integration testing: three different strategies**
  - thread-based testing—integrates the set of classes required to respond to one **input** or event
  - use-based testing—integrates the set of classes required to respond to one **use case**
  - cluster testing—integrates the set of classes required to demonstrate one **collaboration**

22.5 System Testing

System Testing is a level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.



- Usability Testing
- Load Testing
- Regression Testing
- Recovery testing
- Security testing
- Migration testing
- Performance Testing
- Functional Testing
- Deployment Testing
- Hardware/Software Testing

22.5 Regression Testing

- *Regression testing* is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects
  - some bug fix or new features
  - some aspect of the software configuration (its platform or documentation, or the data that support it) is changed.
  - re-executing a subset of all test cases manually or using automated capture/playback tools

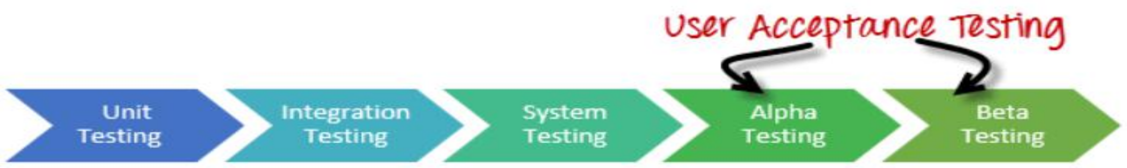
How to select an effective subset from all test cases?

How to priority test cases within limited time?

How to do capture/playback automaticlly ?

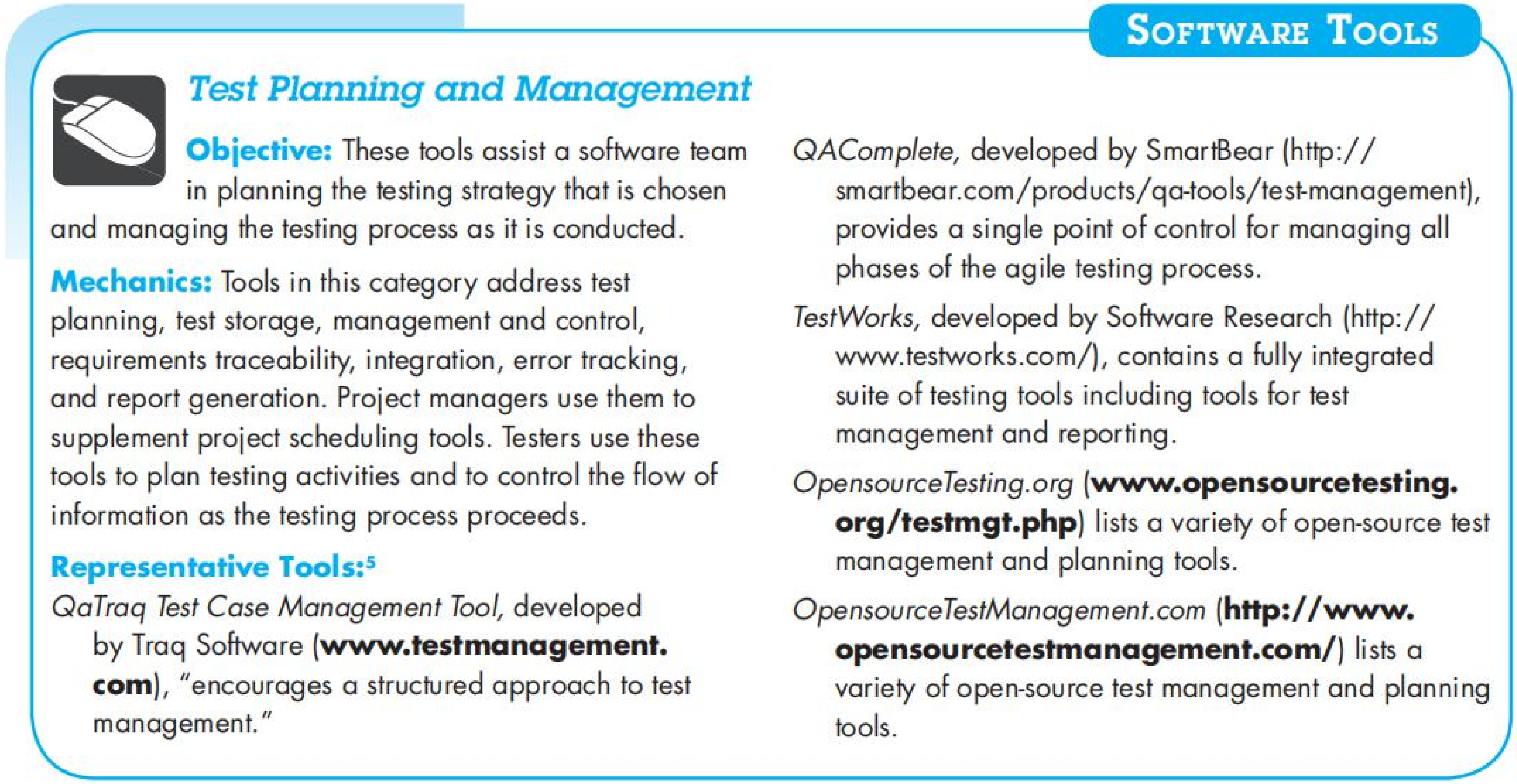
22.7 Validation Testing

- **Validation testing criteria**
  - all functional requirements are satisfied
  - all behavioral characteristics are achieved
  - all content is accurate and properly presented
  - all performance requirements are attained
  - documentation is correct
  - usability and other requirements are met



- **Alpha/Beta testing**
  - Focus is on customer usage
  - Alpha: at developer's site by a representative group of end users
  - Beta: at one or more end-user sites

22.7 Testing management



Software Tools

Test Planning and Management

Objective: These tools assist a software team in planning the testing strategy that is chosen and managing the testing process as it is conducted.

Mechanics: Tools in this category address test planning, test storage, management and control, requirements traceability, integration, error tracking, and report generation. Project managers use them to supplement project scheduling tools. Testers use these tools to plan testing activities and to control the flow of information as the testing process proceeds.

Representative Tools:[5]

QaTraq Test Case Management Tool, developed by Traq Software (**www.testmanagement.com**), "encourages a structured approach to test management."

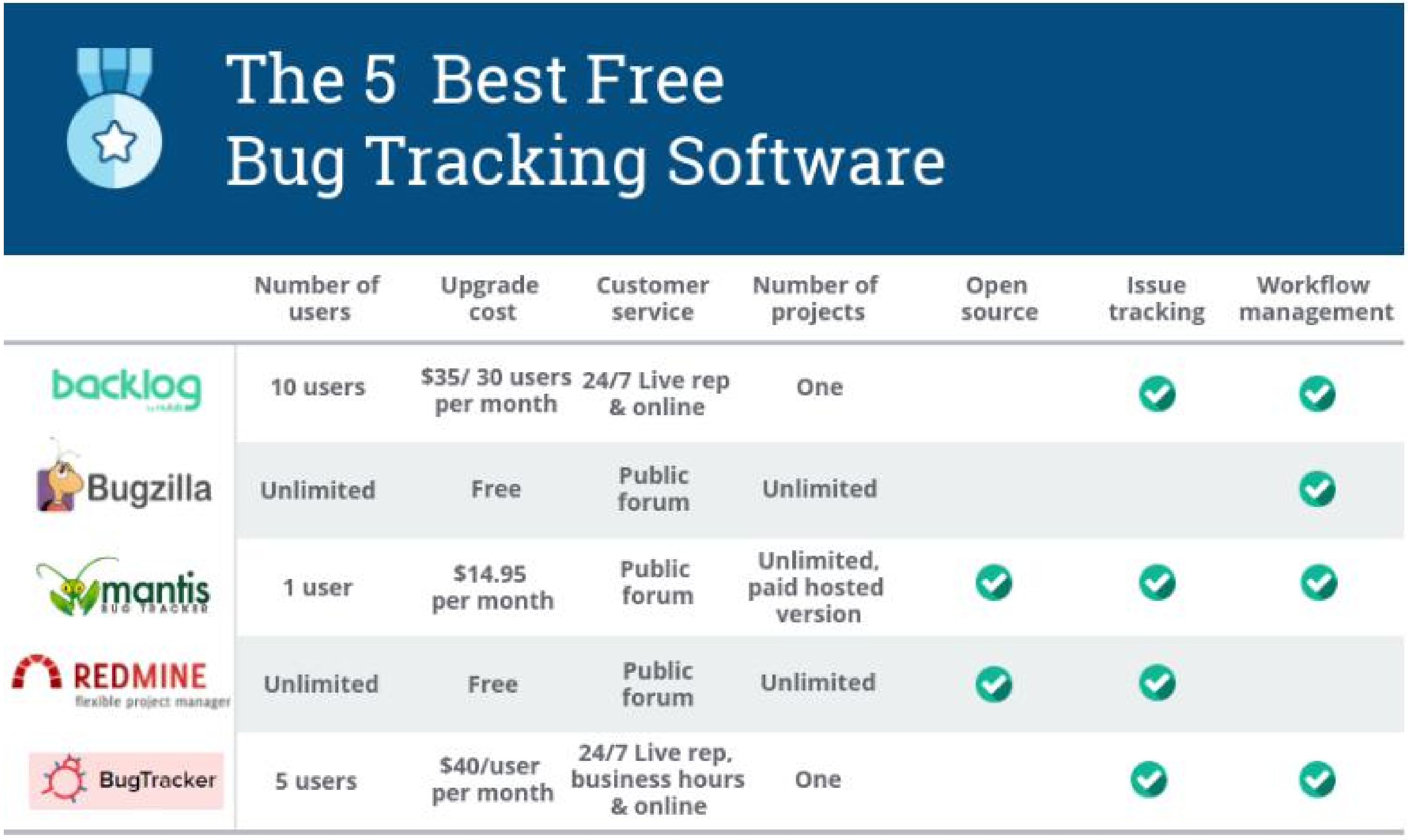
QAComplete, developed by SmartBear (http://smartbear.com/products/qa-tools/test-management), provides a single point of control for managing all phases of the agile testing process.

TestWorks, developed by Software Research (http://www.testworks.com/), contains a fully integrated suite of testing tools including tools for test management and reporting.

OpensourceTesting.org (**www.opensourcetesting.org/testmgt.php**) lists a variety of open-source test management and planning tools.

OpensourceTestManagement.com (**http://www.opensourcetestmanagement.com/**) lists a variety of open-source test management and planning tools.

22.7 Testing management



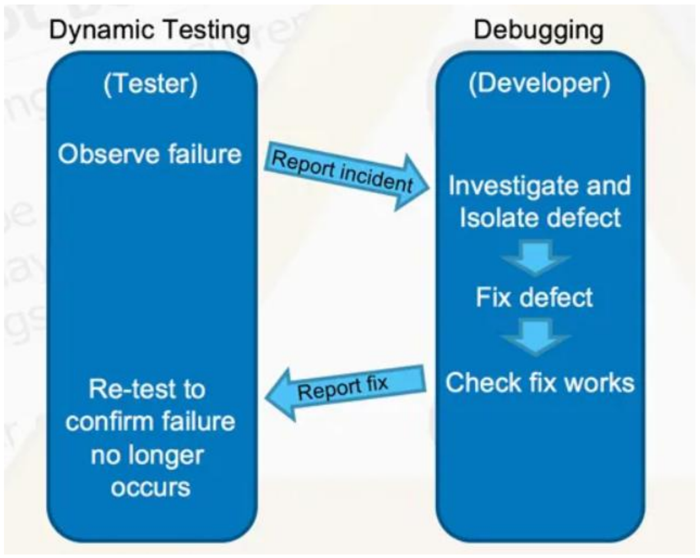
The 5 Best Free Bug Tracking Software

	Number of users	Upgrade cost	Customer service	Number of projects	Open source	Issue tracking	Workflow management
backlog	10 users	$35/ 30 users per month	24/7 Live rep & online	One		✓	✓
Bugzilla	Unlimited	Free	Public forum	Unlimited			✓
mantis BUG TRACKER	1 user	$14.95 per month	Public forum	Unlimited, paid hosted version	✓	✓	✓
REDMINE flexible project manager	Unlimited	Free	Public forum	Unlimited	✓	✓	
BugTracker	5 users	$40/user per month	24/7 Live rep, business hours & online	One		✓	✓

主观题　10分

设置

What is the difference between testing and debugging?



https://dzone.com/articles/the-differences-between-testing-and-debugging

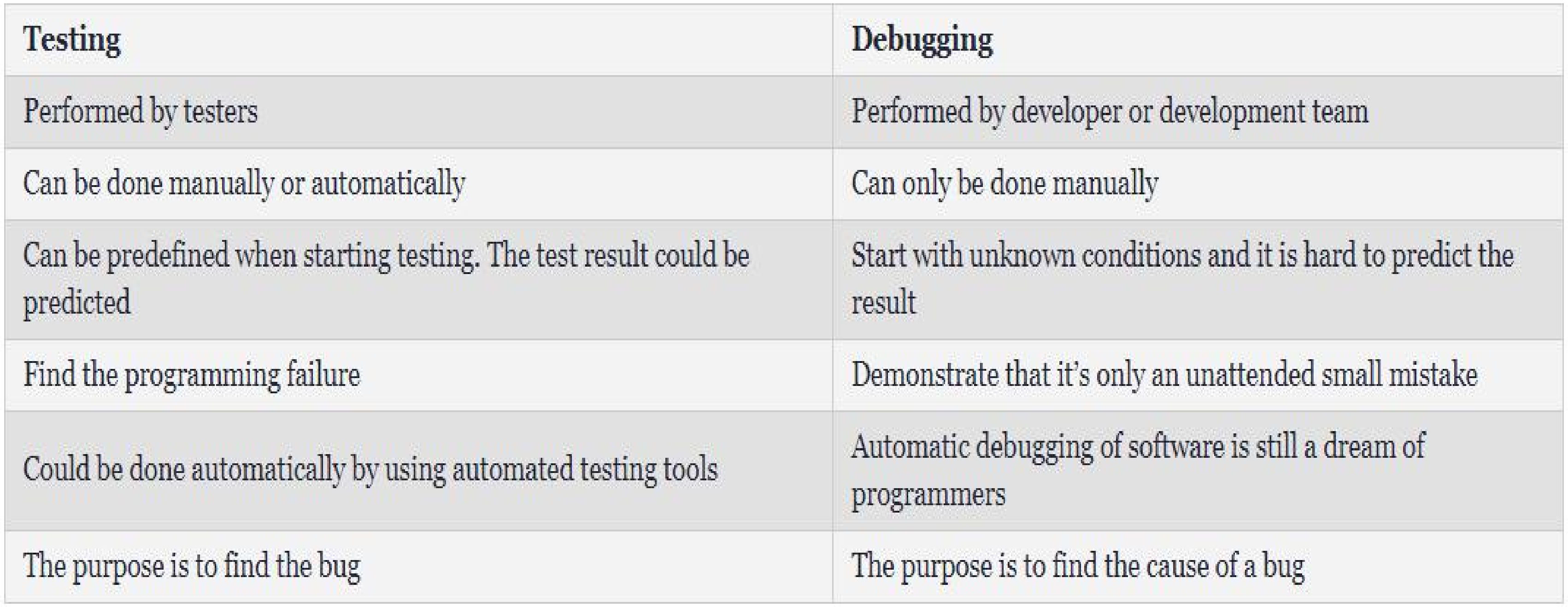
正常使用主观题需2.0以上版本雨课堂

作答

22.9 Debugging: A Diagnostic Process

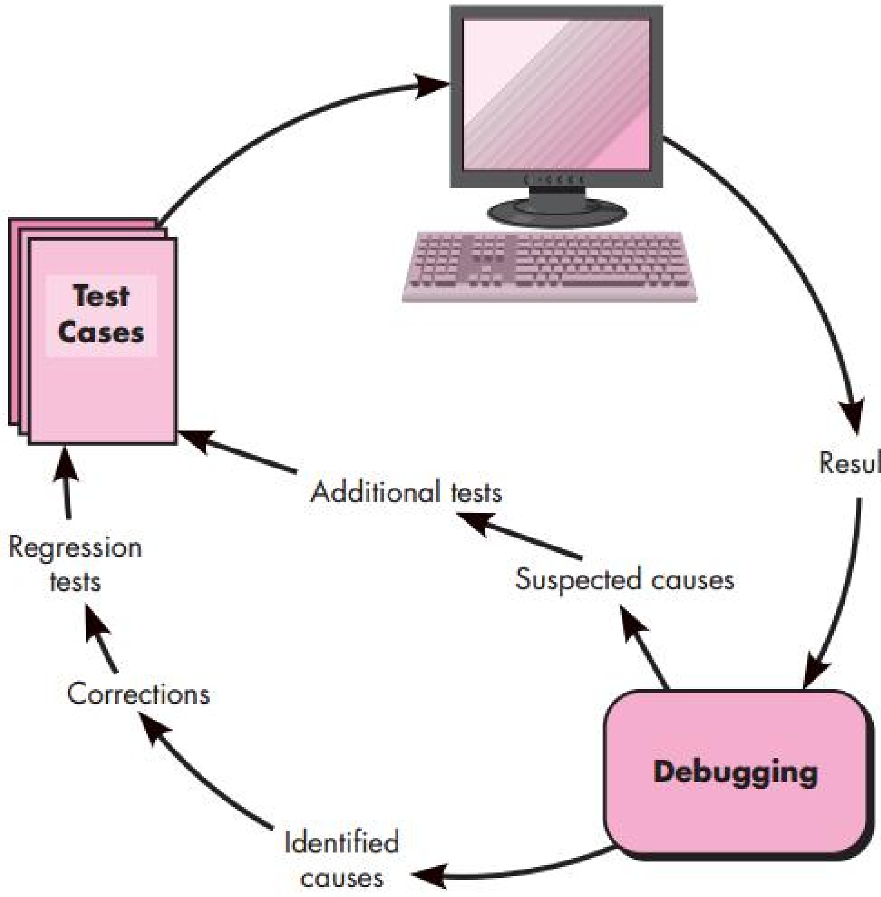
22.1 Strategic Approach

- **Testing and debugging are different activities**

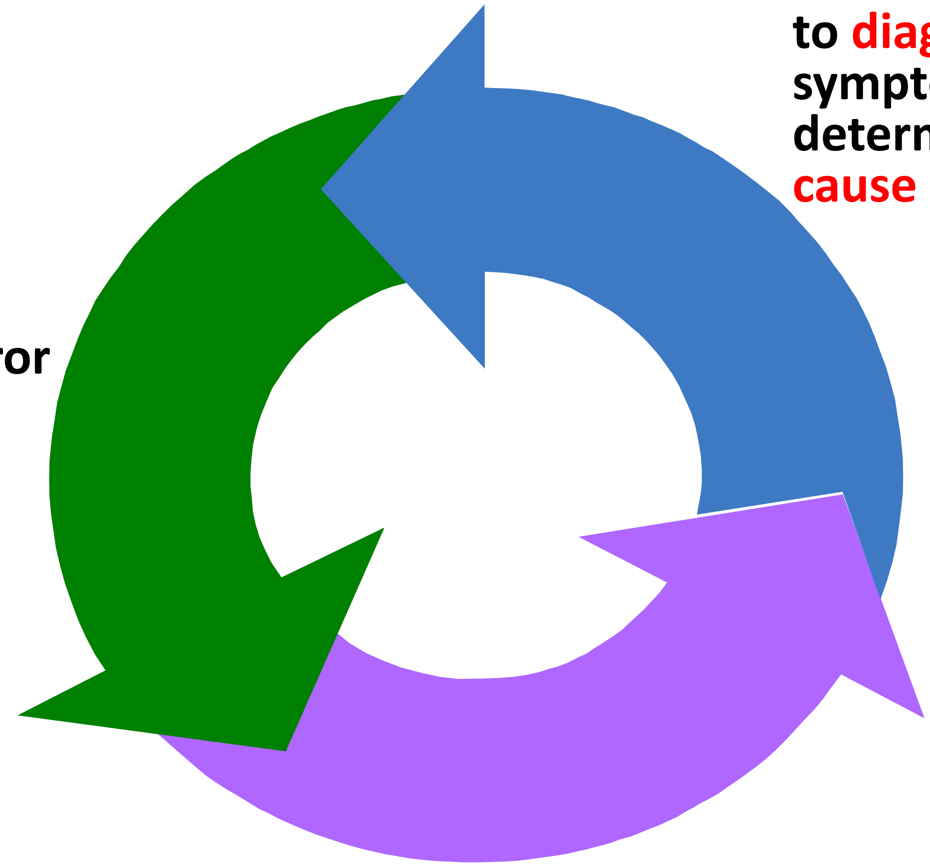


Testing	Debugging
Performed by testers	Performed by developer or development team
Can be done manually or automatically	Can only be done manually
Can be predefined when starting testing. The test result could be predicted	Start with unknown conditions and it is hard to predict the result
Find the programming failure	Demonstrate that it's only an unattended small mistake
Could be done automatically by using automated testing tools	Automatic debugging of software is still a dream of programmers
The purpose is to find the bug	The purpose is to find the cause of a bug

22.9 The Debugging Process



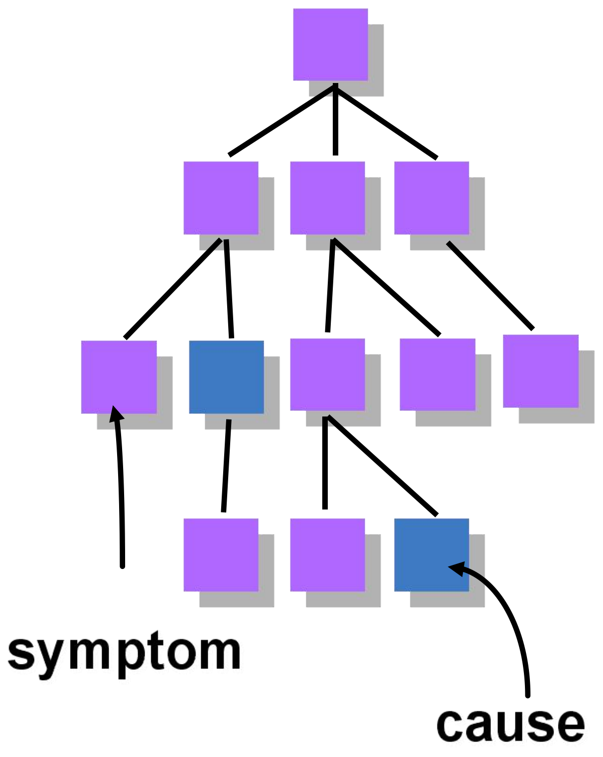
22.9 Debugging Effort



time required
to diagnose the
symptom and
determine the
cause

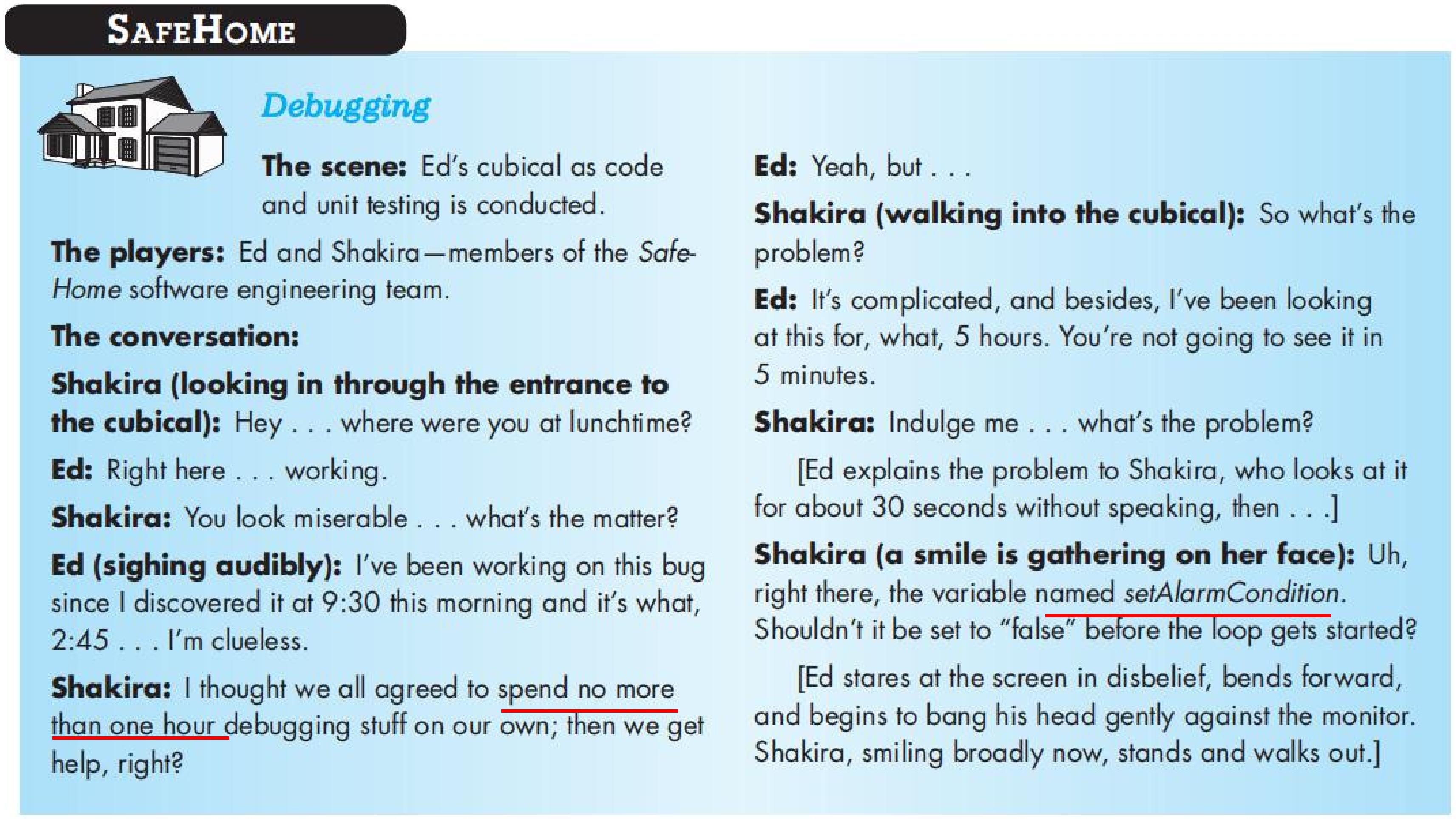
time required
to correct the error
and conduct
regression tests

22.9 Symptoms & Causes



- symptom and cause may be geographically separated
- symptom may disappear when another problem is fixed
- cause may be due to a combination of non-errors
- cause may be due to a system or compiler error
- cause may be due to assumptions that everyone believes
- symptom may be intermittent

22.9 Symptoms & Causes



SafeHome

Debugging

The scene: Ed's cubical as code and unit testing is conducted.

The players: Ed and Shakira—members of the *Safe-Home* software engineering team.

The conversation:

Shakira (looking in through the entrance to the cubical): Hey . . . where were you at lunchtime?

Ed: Right here . . . working.

Shakira: You look miserable . . . what's the matter?

Ed (sighing audibly): I've been working on this bug since I discovered it at 9:30 this morning and it's what, 2:45 . . . I'm clueless.

Shakira: I thought we all agreed to spend no more than one hour debugging stuff on our own; then we get help, right?

Ed: Yeah, but . . .

Shakira (walking into the cubical): So what's the problem?

Ed: It's complicated, and besides, I've been looking at this for, what, 5 hours. You're not going to see it in 5 minutes.

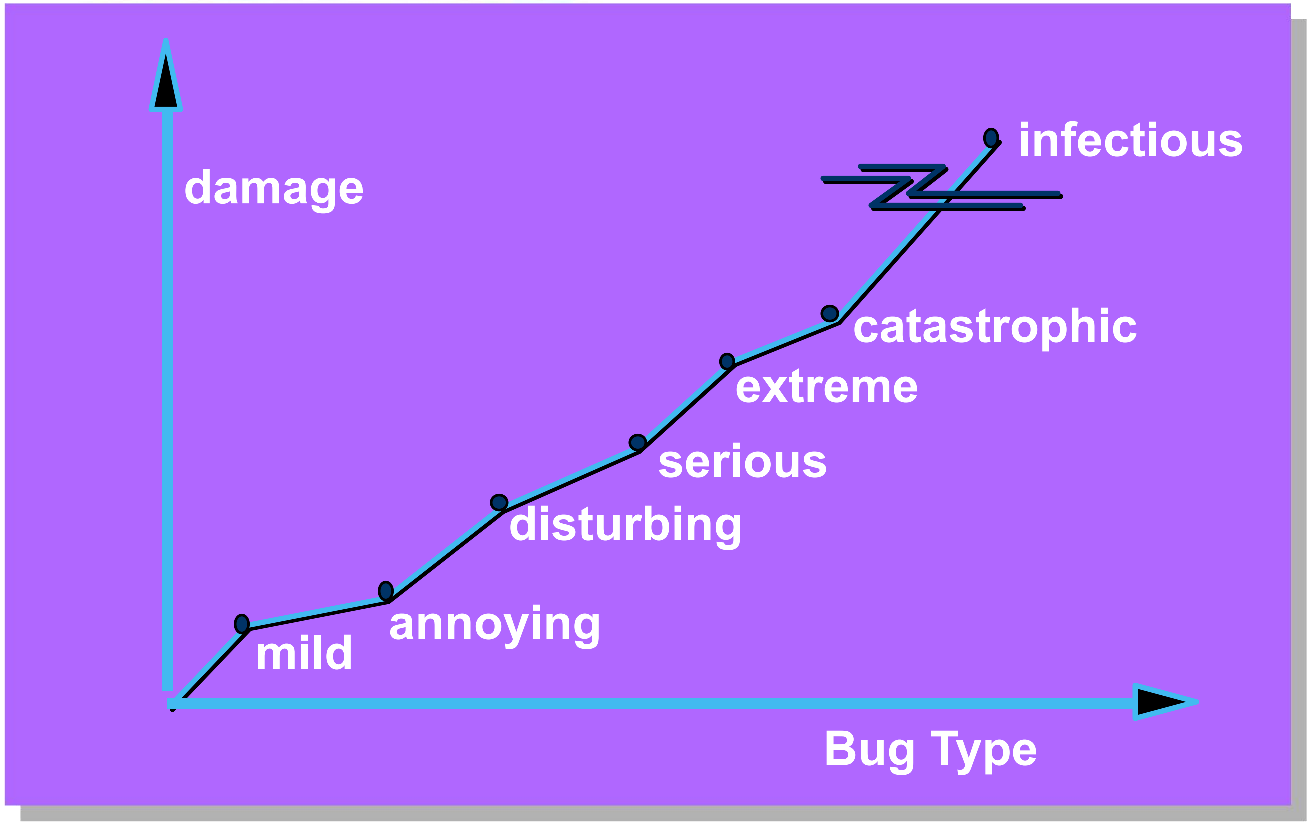
Shakira: Indulge me . . . what's the problem?

[Ed explains the problem to Shakira, who looks at it for about 30 seconds without speaking, then . . .]

Shakira (a smile is gathering on her face): Uh, right there, the variable named *setAlarmCondition*. Shouldn't it be set to "false" before the loop gets started?

[Ed stares at the screen in disbelief, bends forward, and begins to bang his head gently against the monitor. Shakira, smiling broadly now, stands and walks out.]

22.9 Consequences of Bugs



__Bug Categories:__ function-related bugs, system-related bugs, data bugs, coding bugs, design bugs, documentation bugs, standards violations, etc.

22.9 Correcting the Error

- *Is the cause of the bug reproduced in another part of the program?* In many situations, a program defect is caused by an erroneous **pattern** of logic that may be reproduced elsewhere.
- *What "next bug" might be introduced by the fix I'm about to make?* Before the correction is made, the source code (or, better, the design) should be evaluated to assess **coupling** of **logic** and **data structures**.
- *What could we have done to prevent this bug in the first place?* This question is the first step toward establishing a statistical **software quality assurance approach**. If you correct the process as well as the product, the bug will be removed.

22.9 Final Thoughts

- *Think* -- before you act to correct
- Use ***tools*** to gain additional insight
- Once you correct the bug, use ***regression*** testing to uncover any side *effects*

Summary

- **Testing concept**
  - ✓ strategic
  - ✓ V & V
  - ✓ testing plan
  - ✓ test suite & test case
- **Unit testing and Integration testing**
- **System testing**
- **Validation testing**
- **Debugging**

THE END