中山大學
SUN YAT-SEN UNIVERSITY

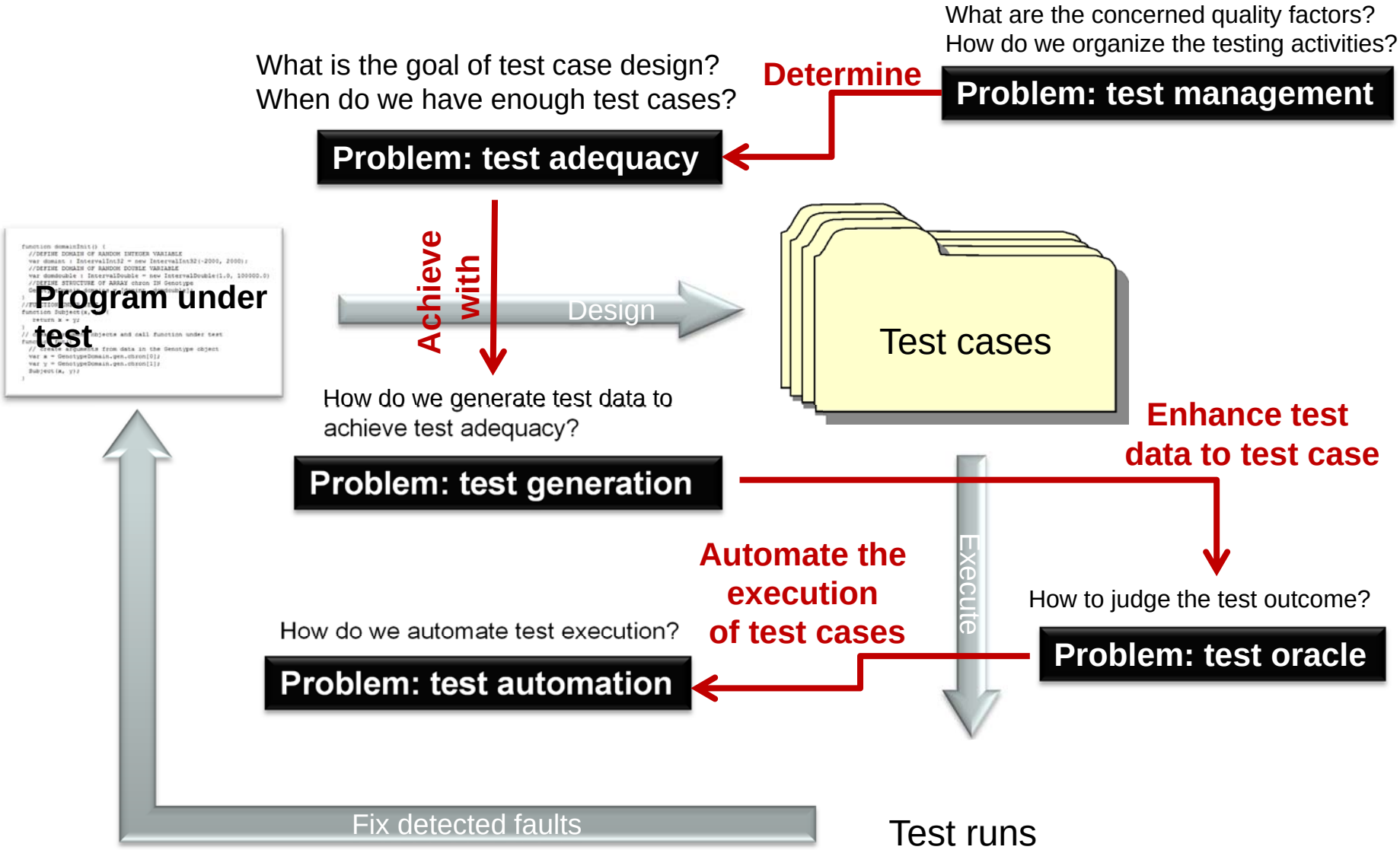
Part II [Problem]
7. Test Management

SE-307 Software Testing Techniques

http://my.ss.sysu.edu.cn/wiki/display/SE307/Home

Instructor: Dr. Wang Xinming, School of Software, Sun Yat-Sen University

Review: 5 problems in testing



Review: GUI testing

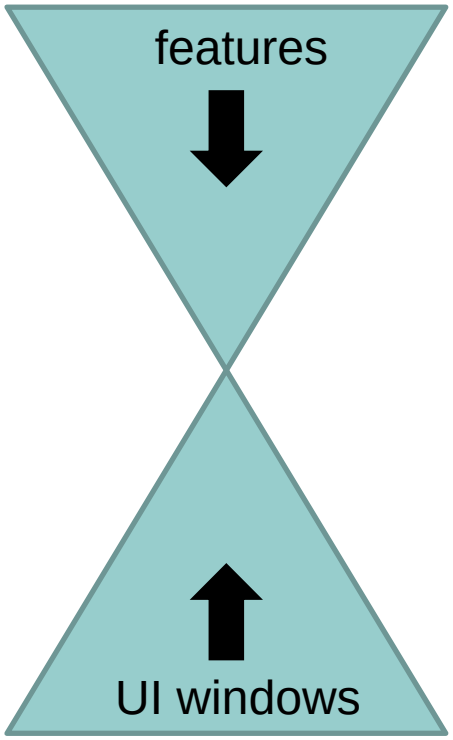
- Event-based test adequacy
  - **Step 1**: enumerate all windows of the program and build the window hierarchy.
  - **Step 2**: for each window, identify the events that can occur and describe them with event operators.
  - **Step 3**: design test cases to cover every representative events.
  - **Step 4:** identify the “follow” relation between events and build the event flow graph.
  - **Step 5**: design test cases to cover intra-window event interactions.
  - **Step 6**: design test cases to cover inter-window event interactions.

Review: GUI testing

- State-based test adequacy
  - **Step 1**: analyze the requirement, identify key features to test.
  - **Step 2**: for each feature, enumerate key scenarios.
  - **Step 3**: construct the finite state model for each feature using UML state chart.
  - **Step 4:** design test cases to traverse the state/transition.

Review: GUI testing

State-based testing

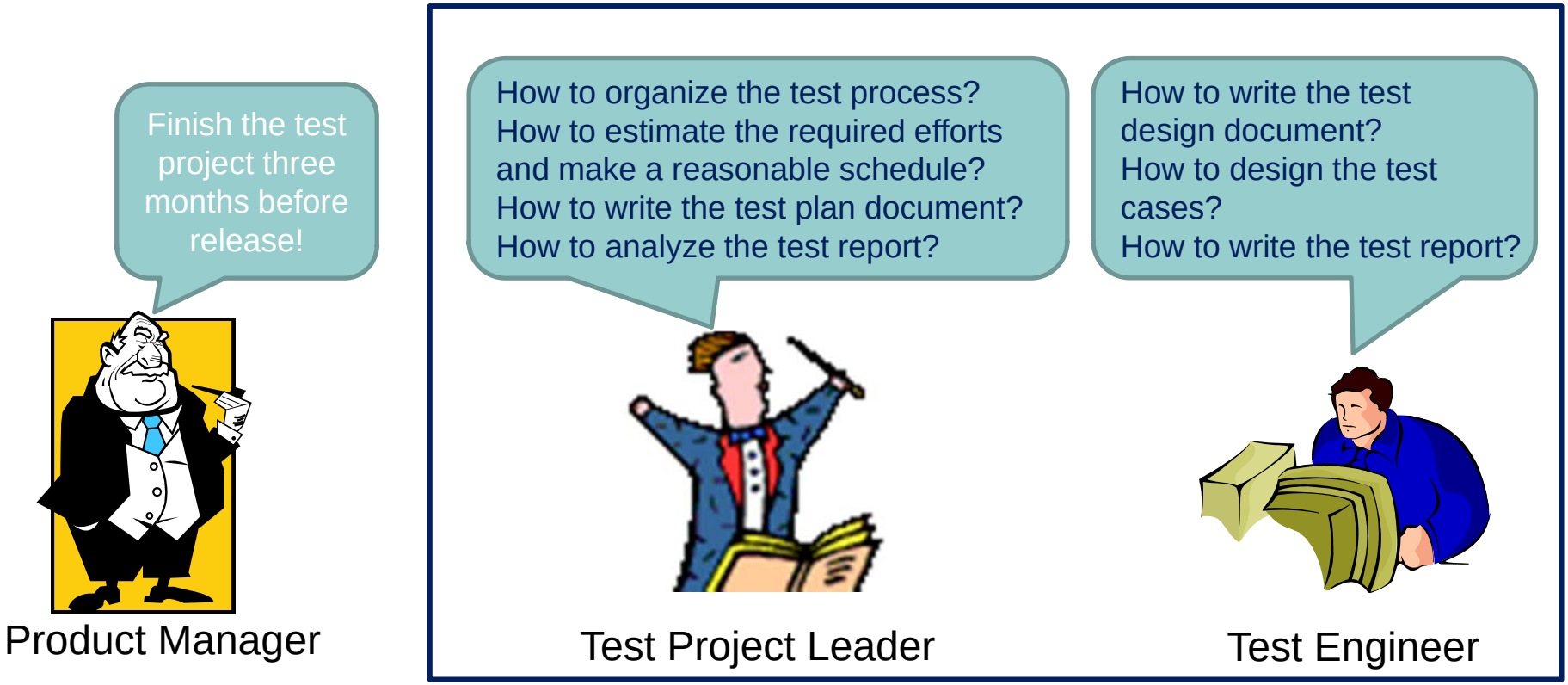


Event-based testing

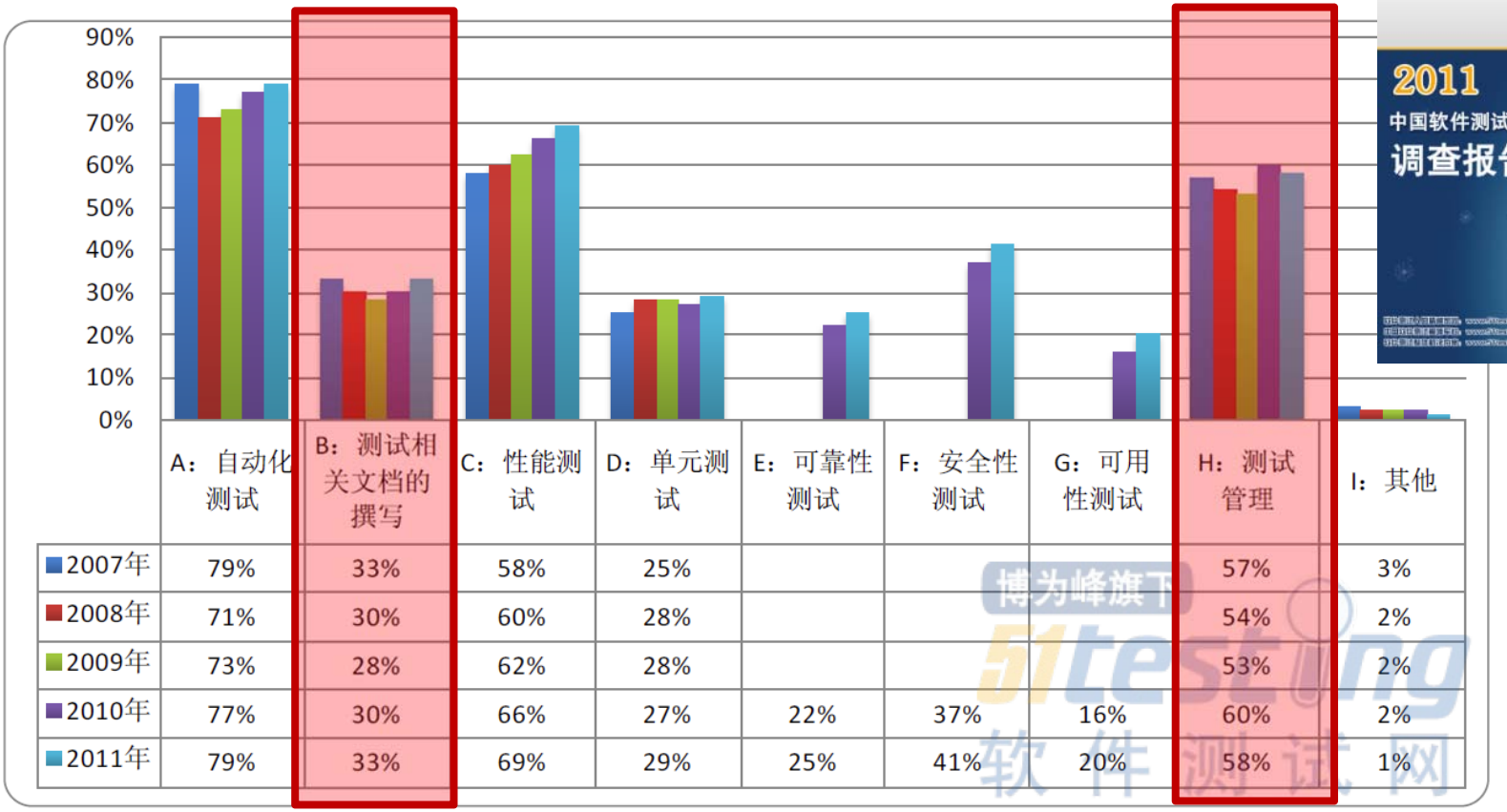
	Event-based GUI testing	State-based GUI testing
Information source	UI design	Requirement
Focus	Low-level GUI events and their interaction	High-level software features
Key artifact	Event operators and their interaction table	State-transition diagram
Characteristics of test cases	Short operation sequences that expose errors in GUI, such as: • Unexpected parameter values • Unexpected event interaction	Long operation sequences that expose errors in application logic, such as: • Omissions in functionality • Incorrect transitions

What is Test Management Problem?

- **Test management** is the activity of managing the test project
  - How to manage the ***test process***? （过程）
  - How to write ***test document*** to facilitate test process?（文档）



Is Test Management Important?



	A：自动化测试	B：测试相关文档的撰写	C：性能测试	D：单元测试	E：可靠性测试	F：安全性测试	G：可用性测试	H：测试管理	I：其他
2007年	79%	33%	58%	25%				57%	3%
2008年	71%	30%	60%	28%				54%	2%
2009年	73%	28%	62%	28%				53%	2%
2010年	77%	30%	66%	27%	22%	37%	16%	60%	2%
2011年	79%	33%	69%	29%	25%	41%	20%	58%	1%

历届调查中软件测试从业人员希望提高的软件测试技能

测试经理：小李，这周五之前把测试设计文档写好给我，下周完成设计用例的开发工作，6月4号之前把测试报告发给我。

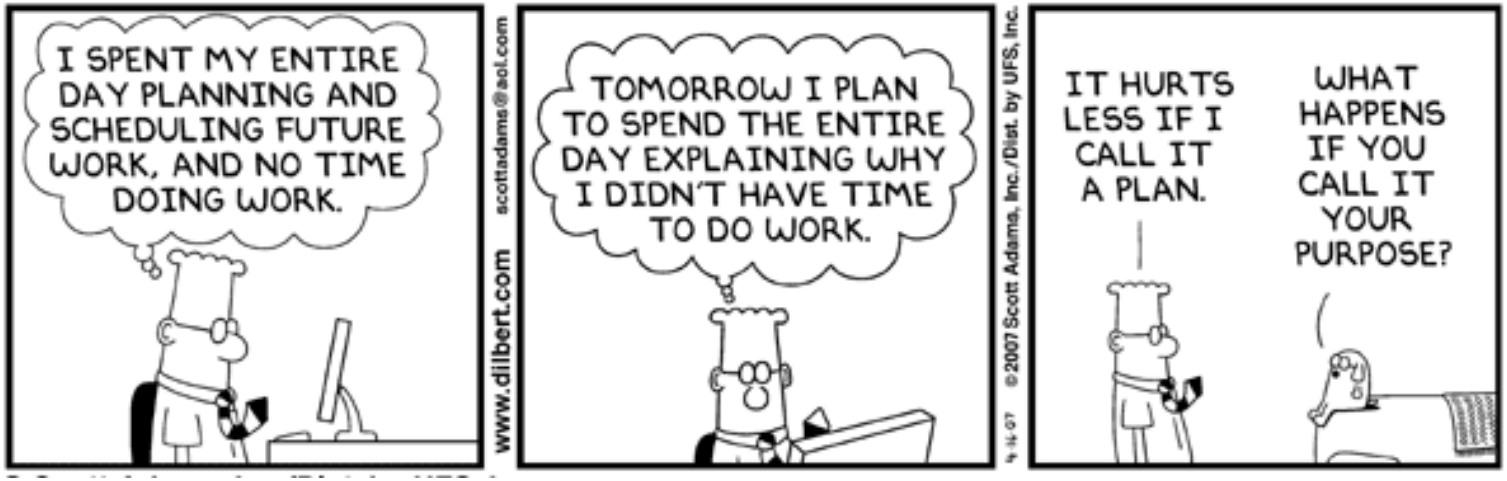
How to finish these tasks?

Why it is Important?

- **Fact 1**: managers like employers who know how to write documents.
  - Documentation supports process visibility
    - For internal use (schedule tracking)
    - For external authorities (certification, auditing)
  - Document can be inspected to improve the process
    - Maintain a body of knowledge reused across projects
    - Summarize and present data for process improvement
  - Increase reusability of artifacts within and across projects

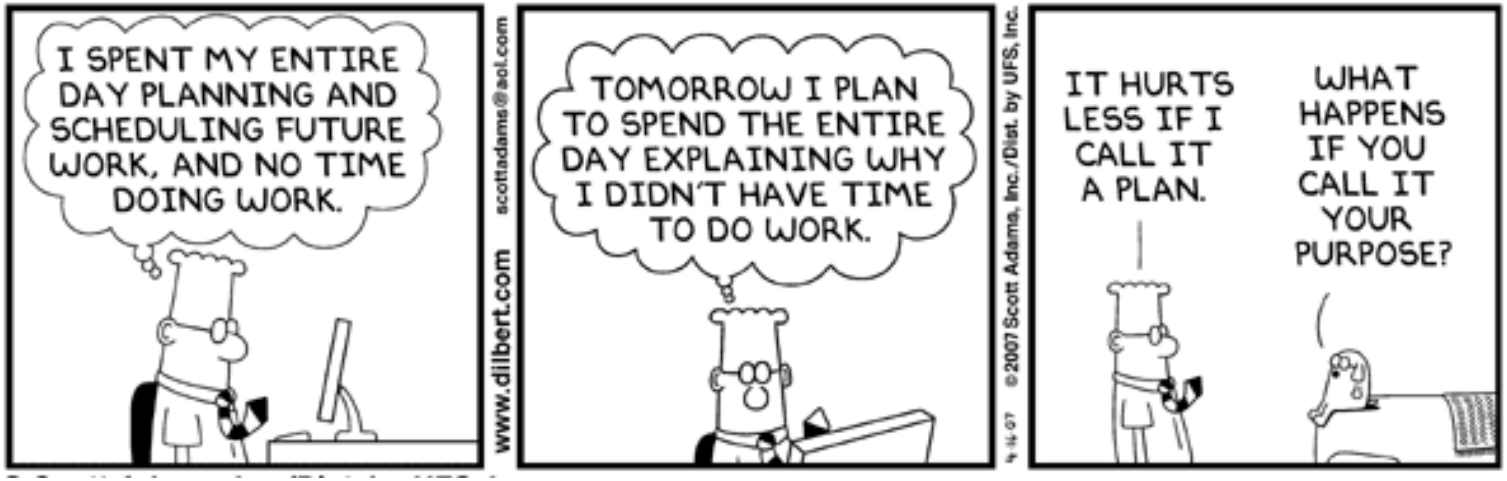
Why it is Hard?

- **Fact 2**: developers hate documentation.



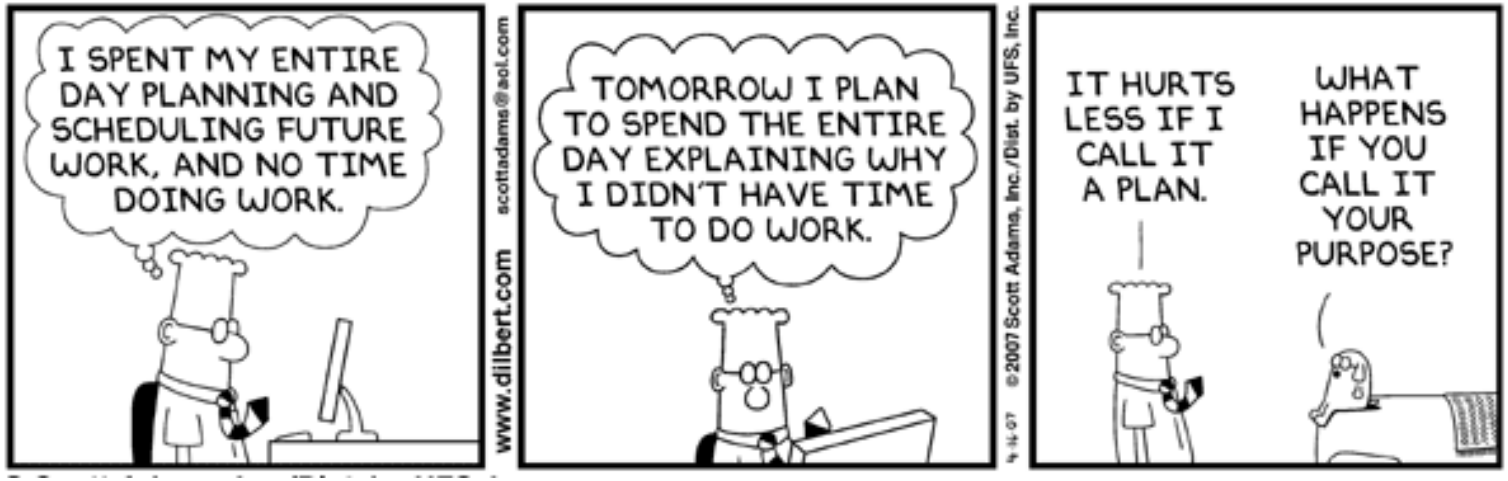
Why it is Hard?

- **Fact 2**: developers hate documentation.
  - They don't appreciate the usefulness of documents.



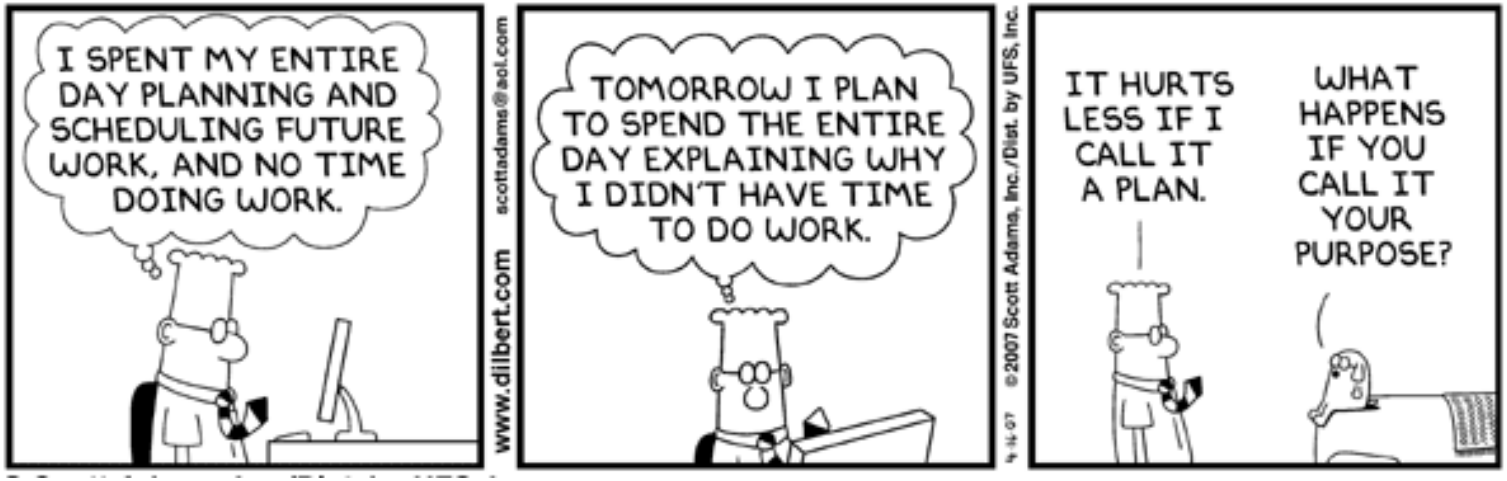
Why it is Hard?

- **Fact 2**: developers hate documentation.
  - They don’t appreciate the usefulness of documents.
  - They prefer writing code to writing documents.



Why it is Hard?

- **Fact 2**: developers hate documentation.
  - They don’t appreciate the usefulness of documents.
  - They prefer writing code to writing documents.
  - They don’t know how to write good documents.



Some General Tips

- General Rules for Writing Good Documents
  - Know your target
  - Keep it short
  - Mind your own business
  - Use clear terminology
  - Kill redundancy
  - Use precise language
  - Explain your reasoning

Rule #1: Know Your Target

- Who are the most important target of your document?
  - Managers? Programmers? Test engineers? Customer? Technical writers?
  - Always keep your intended audience in mind and understand what they want.
  - When in doubt, don’t shy away from asking them.

Rule #2: Keep it Short

- Short documents are:
  - Easier to read
  - Easier to write
  - Easier to maintain and keep up-to-date
  - Less likely to be contradictory
  - More likely to be simple designs
- How to keep it short:
  - Kill empty sections
  - Kill ‘cut and paste’ stuff
  - Kill unnecessary text of obvious systems
- Remember:
  - Developers and test engineers like bullet list and check list.
  - Customers like readable stories.
  - Managers like statistics.

Rule #3: Mind Your Own Business

- Don't tell other how to do their jobs.
  - People hate that.

This is not your business.

These test scripts may be implemented using a 'for' loop that read from a database file that keep all the usernames and passwords.

This is your business.

Test automation considerations:

Username consist of 8 characters.
Password consist of 12 digits.
A set of pre-defined users are defined in a database file.

Rule #4 Use Clear Terminology

- Use plain English
- Avoid weasel words
  - We will use a business-led project management process to optimize our strategic core issues and create disruptive innovations that redefine the market.
- Avoid company-specific terms
- Use new terms consistently
- Must have a glossary for acronyms

Rule #5 Reduce Redundancy

- Duplication is the devil, leads to confusion, update errors.

"CombatStats.doc"

Strength increases the player's damage by STRENGTH/2.
Dexterity increases the player's accuracy by DEXTERITY/3

"Items.doc"

Strength increases the player's damage by STRENGTH/2.
Dexterity increases the player's accuracy by DEXTERITY/3

Wrong!

"CombatStats.doc"

Strength increases the player's damage by STRENGTH/2.
Dexterity increases the player's accuracy by DEXTERITY/3

"Items.doc"

Enchantments on an item can increase the players stats when worn. See CombatStats.doc for more details.

Better

Rule #6: Use Precise Language

- Use strong, declarative language
- No ‘maybe’, ‘could’, ‘might’
- Even avoid ‘may’.
- Don’t be ambiguous
- Don’t say ‘we’
- Move ‘maybe’ features to later phases.

Rule #6: Use Precise Language

We might try to cover all statements for some of the most important classes.

In the future, we may cover the du-paths for these classes, if time allowed.

In order to find test data that cover the du-paths, maybe we can use a third party test generation tool.

Wrong!

Rule #6: Use Precise Language

In phase 1, the test engineers must design test cases to cover all statements for all classes in package sysu.apple and sysu.banana.

If phase 2 is carried out, the test engineers must design test cases to cover all the du-paths for all classes in package sysu.apple and sysu.banana.

Whether phase 2 is carried out depends on whether a third-party test generation tool can be obtained by the testing team. Candidates include Microsoft Pex and Agitar One.

Better

Rule #7: Explain Your Reasoning

- Capturing your reasoning is especially useful for longer projects, where the team may literally forget why they chose one side or the other.

- Capturing your reasoning, by extension, reduces the number of times contentious issues are reopened.

Process vs. Documentation

- Process management and documentation: which aspect shall we put more emphasis?
  - **Point of view 1**: process management is more important. Test engineer shall focus on thinking the overall test process. They don't need to write very detailed documents with tedious format requirement.
  - **Point of view 2**: documentation is more important. Test engineer shall put more emphasis on writing documents as detailed as possible, and the document must fully comply with standard format such as IEEE 829.

- My point of view:
  - Over-emphasizing documentation kill creativity, but overlooking documentation might put your project at risk.
  - Use process management tool (such as testlink) to reduce the burden of formatting.

Understanding Test Management

- Test Process Overview

- IEEE 829 Software Test Documentation Standard

Understanding Test Management

- Test Process Overview

- IEEE 829 Software Test Documentation Standard

Test Process

- **Test process**: the process which defines procedures and activities required to successfully complete the testing a project.

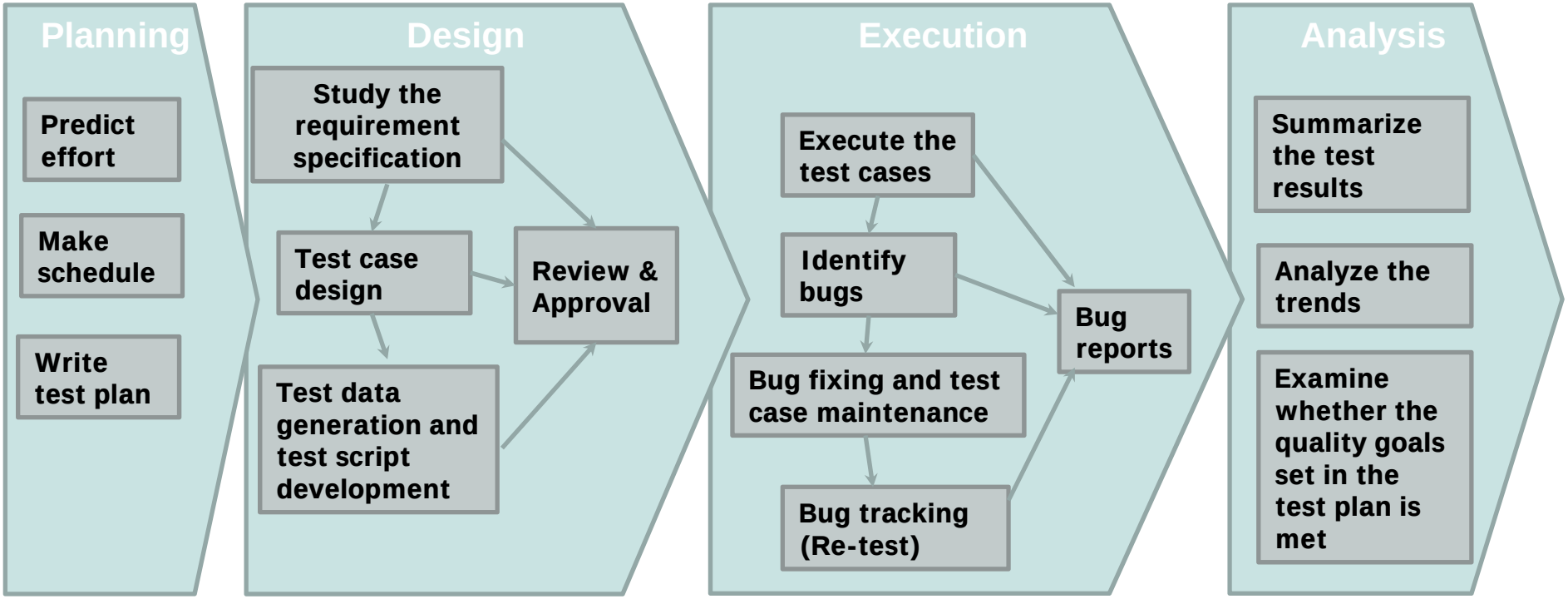
- **Understanding test process**:
  - **What**: What are the main activities?
  - **When**: When is each activity carried out?
  - **Who**: Who carry out each activity?
  - **How**: How is each activity done?

Activities in Test Process (1)

- Divided by levels: **Unit**, **Integration**, **System**, **Acceptance**
- **Unit Testing** (or Component Testing)
  - Testing at the code module level.
  - Typically performed by developers exclusively.
- **Integration Testing**
  - Testing to assure that the code modules and groups of code modules (sub-systems) work correctly with each other.
  - Lower, module-to-module level is typically performed by developers. Higher, subsystem-to-subsystem level is typically performed by test engineers.
- **System Testing**
  - Testing of the whole system outside of the production environment.
  - Typically performed by test engineers.
- **Acceptance Testing**
  - Testing to verify that the system meets the user's requirement.
  - "Alpha testing": performed by the users in the development environment.
  - "Beta testing": performed by the users in the customer environment.

Activities in Test Process (2)

- Divided by phases: **Planning**, **Design**, **Execution**, **Analysis**



Activities in The Test Process

	Planning	Design	Execution	Analysis
Unit testing	When? Who? How?	When? Who? How?	When? Who? How?	When? Who? How?
Integration testing	When? Who? How?	When? Who? How?	When? Who? How?	When? Who? How?
System testing	When? Who? How?	When? Who? How?	When? Who? How?	When? Who? How?
Acceptance testing	When? Who? How?	When? Who? How?	When? Who? How?	When? Who? How?

Activities in The Test Process

	Planning	Design	Execution	Analysis
Unit testing	Who? Developers	Who? Developers	Who? Developers	Who? Developers
Integration testing	Who? Developers or Test Engineer	Who? Developers or Test Engineer	Who? Developers or Test Engineer	Who? Developers or Test Engineer
System testing	Who? Test Engineers	Who? Test Engineers	Who? Test Engineers	Who? Test Engineers
Acceptance testing	Who? End-Users	Who? End-Users	Who? End-Users	Who? End-Users

When to Carry Out Each Activity?

- Testing activities are carried out in software development lifecycle.
  - But exactly when? And how do test activities interleave with other development activities?

The devil's way: all test activities are carried out after the software has been deployed (and done by the end-users, actually).

Software Development Lifecycles

- The process used to create a software product from its initial conception to its public release is known as the software development lifecycle model.
- There are four frequently used models, with most others just variations of these:
  - **Code–and–Fix**
  - **Waterfall**
  - **Spiral**
  - **Agile**

Code-and-Fix Model

- The product is implemented without requirements or specifications, or any attempt at design.
- The developers simply throw code together and rework it as many times as necessary to satisfy the client.
- It is used in small project and is not scalable to serious software products of any reasonable size.

Testing with Code-and-Fix Model

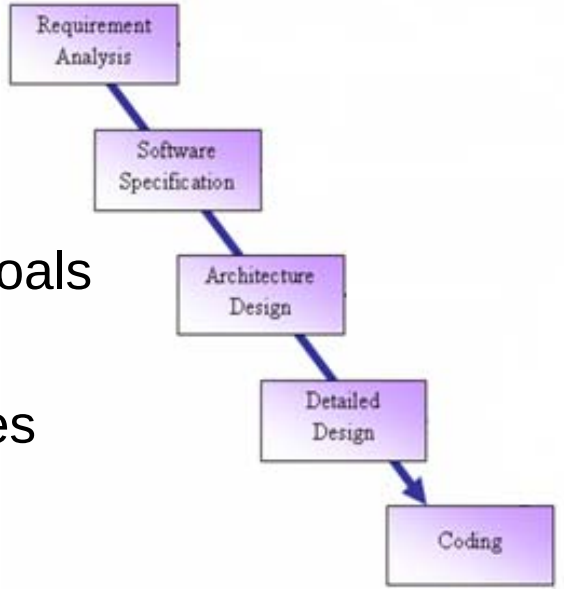
- A test engineer on a code-and-fix project feels like…

- It is a mess: if no one in the team knows exactly *what the software is expected to do* and *how it is implemented*, how can you test?

- And what is the point of analyzing the test results?

Waterfall Model

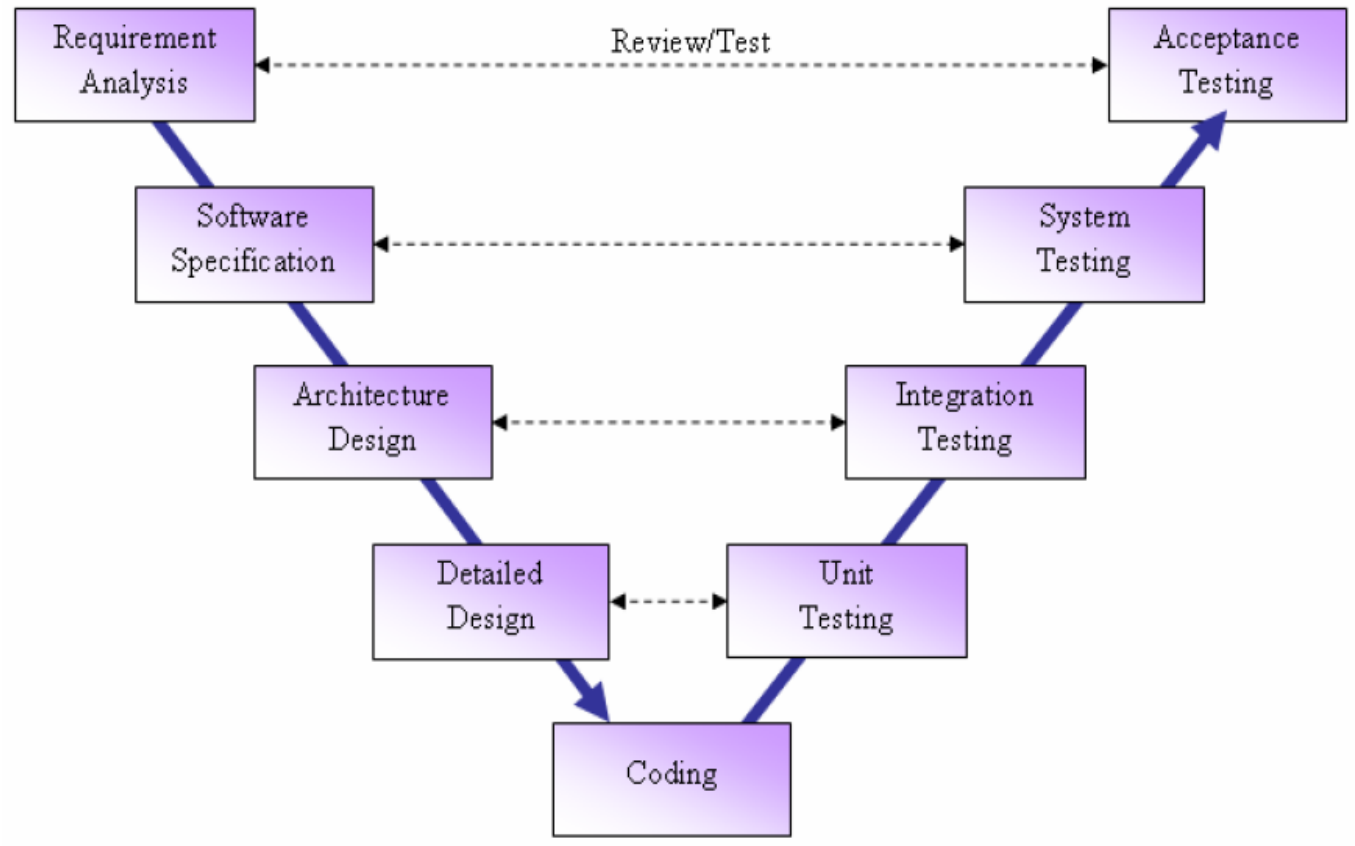
- A linear life-cycle model
- Benefits?
  - Formal, standard; specific phases with clear goals
  - Clear divisions between phases
  - Good feedback loops between adjacent phases



- Drawbacks?
  - Assumes requirements will be clear and well-understood
  - Requires a lot of planning up front (not always easy)
  - Rigid, linear; not adaptable to change in the product
  - Costly to "swim upstream" back to a previous phase
  - Nothing to show until almost done ("we're 90% done, I swear!")

Testing in Waterfall Model

- Test activities in the waterfall model are organized as:

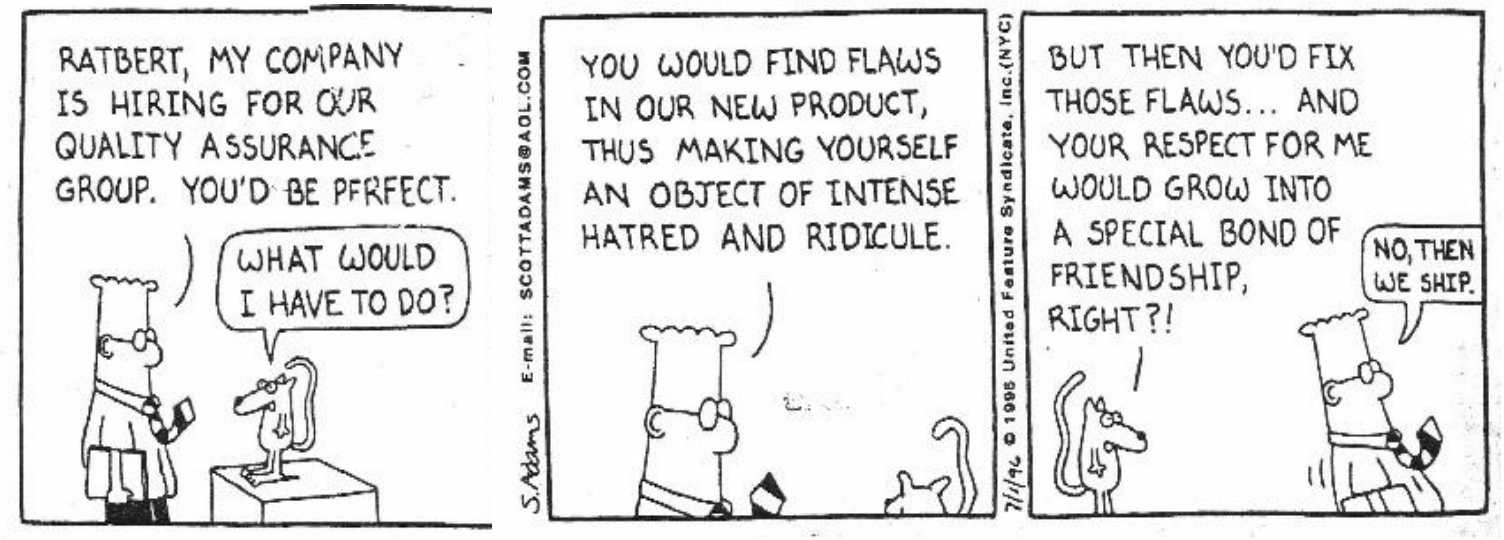


When to Carry Out Test Activities

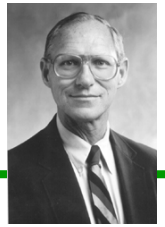
	Planning	Preparation	Execution	Analysis
Unit testing	When? After coding	When? After unit test plan is completed	When? After unit test design is completed.	When? After unit test has been executed.
Integration testing	When? After unit testing completes	When? After integration test plan is completed	When? After integration test design is completed.	When? After integration test has been executed.
System testing	When? After integration testing completes	When? After system test plan is completed	When? After system test design is completed.	When? After system test has been executed.
Acceptance testing	When? After system testing completes	When? After acceptance test plan is completed	When? After acceptance test design is completed.	When? After acceptance test has been executed.

Pros and Cons

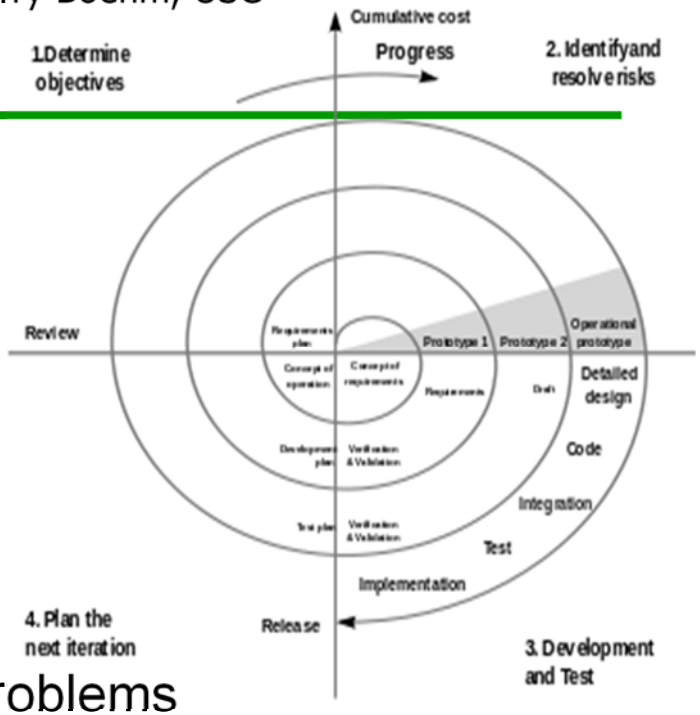
- Pros:
  - Test engineer know exactly what the software is expected to do, so that there's no question about whether something is a feature or a bug.
- Cons:
  - Because testing occurs only at the end, fundamental problems could creep in early on and not be detected until days before the scheduled product release, and it is too late.
  - You are the one last stop before the software can be released. But you keep finding problems. They seem endless. You are hated by everyone.



Spiral Model



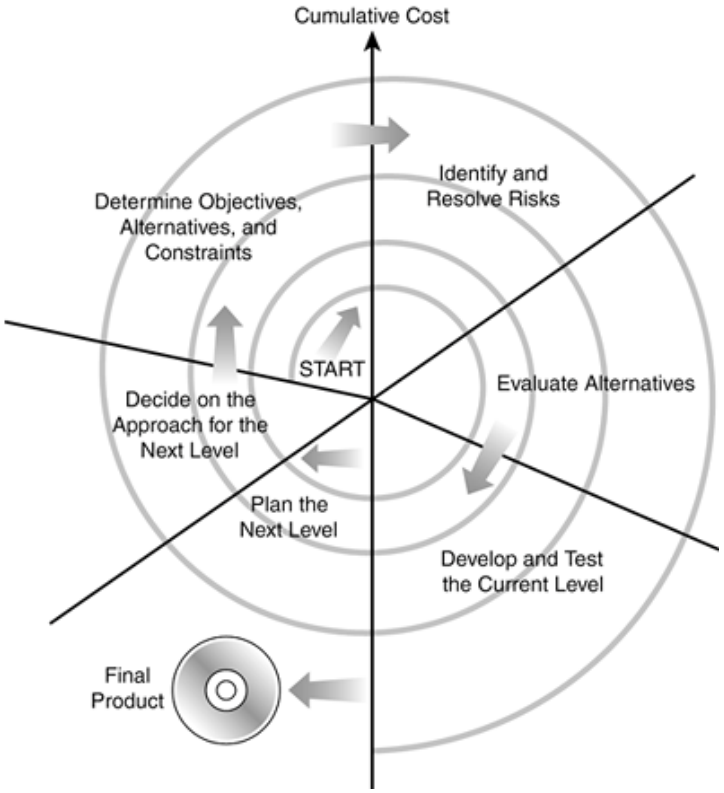
Barry Boehm, USC



- Steps taken at each loop:
  - Determine objectives
  - Identify and resolve risks
  - Development and test
  - Plan the next iteration
- Benefits?
  - Provides early indication of unforeseen problems
  - Always addresses the biggest risk first
  - Accommodates changes, growth
  - Eliminates errors and unattractive choices early
- Drawbacks?
  - Relies on developers to have risk-assessment expertise
  - Over-focuses on risk and "putting out fires"; other features may go ignored because they are not "risky" enough

Testing in Spiral Model

- Similar to testing in waterfall model, except that test engineers now get a chance to influence the product.
- The test result of the present iteration will help identifying and resolve risks at the next iteration.



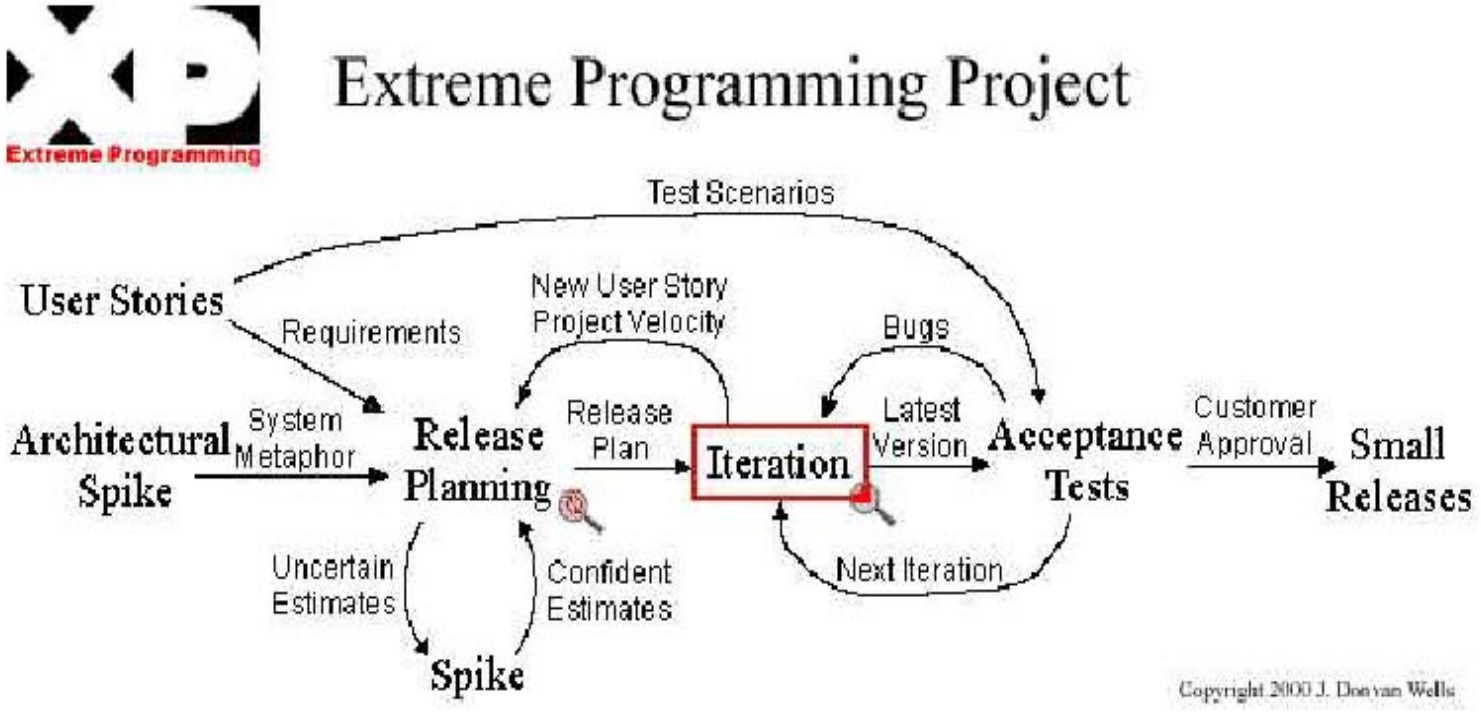
Agile Model

- **Agile development model** is a conceptual framework for software engineering that promotes development iterations throughout the life-cycle of the project.
- Software developed during one unit of time is referred to as an iteration, which may last from one to four weeks.
- Agile methods also emphasize working software as the primary measure of progress
- **Characteristics:**
  - Light Weighted methodology
  - Small to medium sized teams
  - Vague and/or changing requirements and techniques
  - Simple design
  - Minimal system into production

The Most Misunderstood Model

Extreme Programming

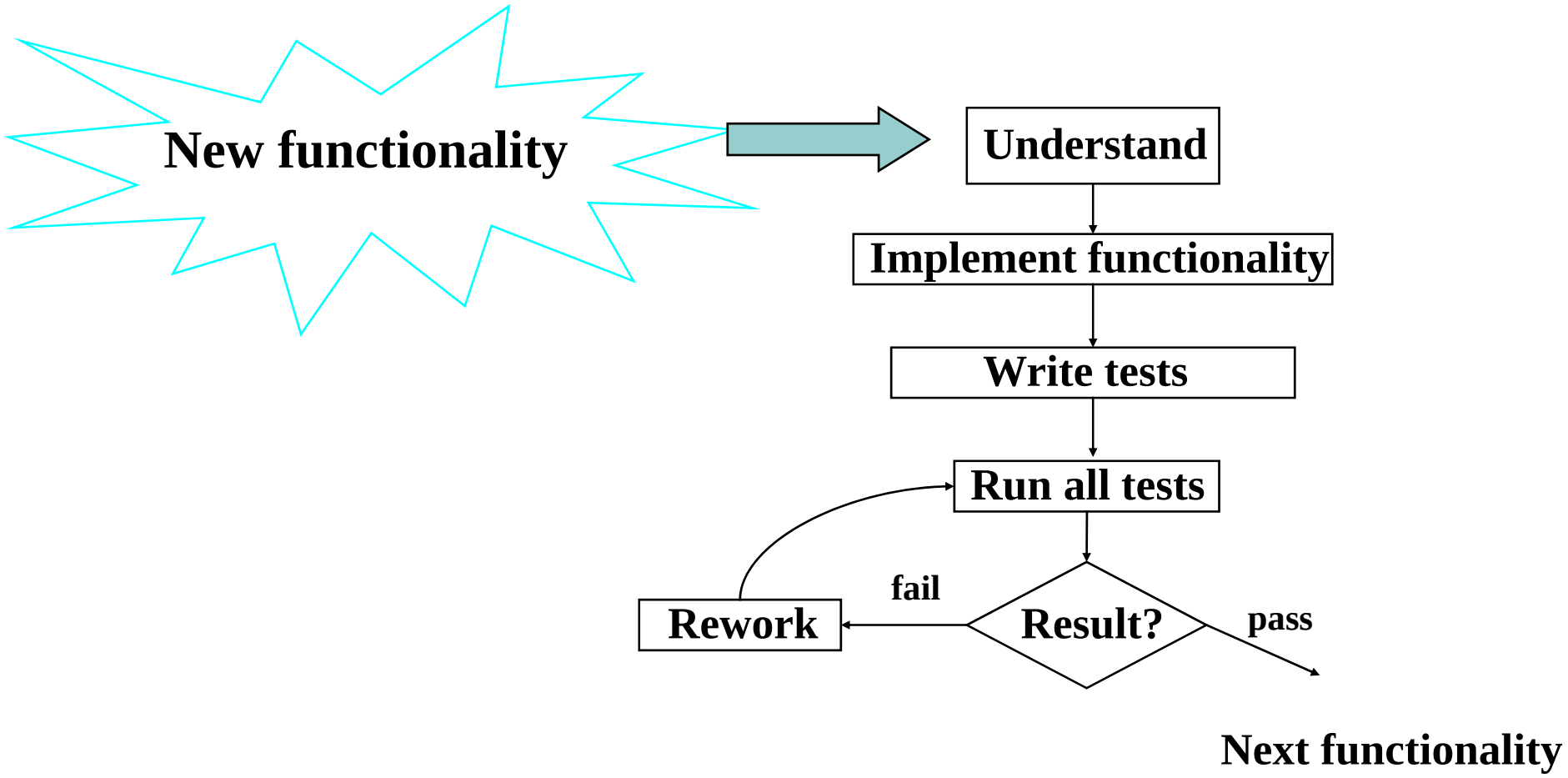
- Most prominent agile development method
- Prescribes a set of daily stakeholder practices
- “Extreme” levels of practicing leads to more responsive software.



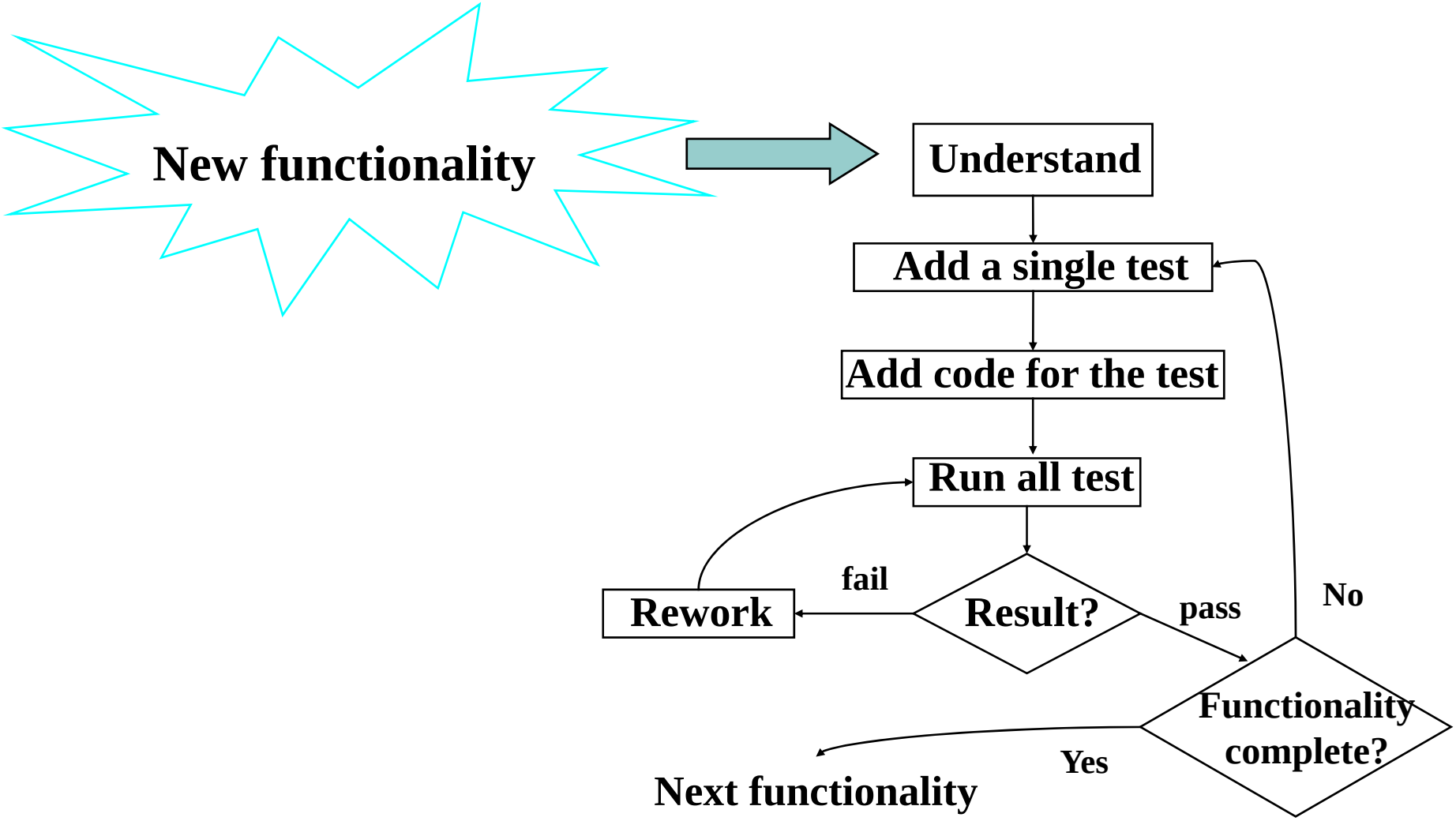
Test Driven Development (TDD)

- TDD was introduced as part of Extreme Programming.
- Test-driven development (TDD) is an approach to program development in which you inter-leave testing and code development.
- Tests are written before code and ‘passing’ the tests is the critical driver of development.
- You develop code incrementally, along with a test for that increment. You don’t move on to the next increment until the code that you have developed passes its test.
- Usually used for developer testing, but can also be used for independent testing (ATDD).

The Normal Way



Test-Driven Development



Advantages of TDD

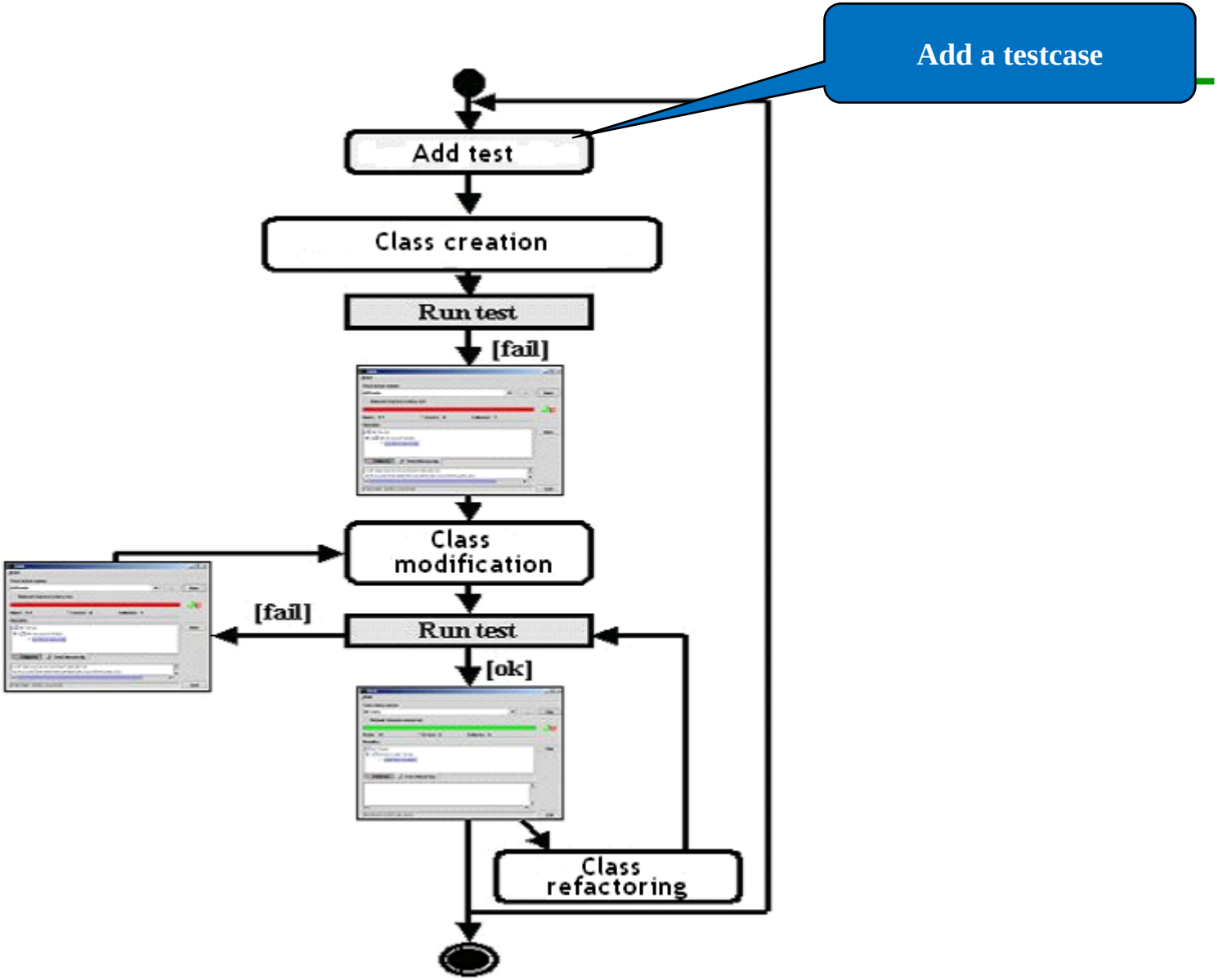
- Each method has associated test cases
  - the confidence of our code increases
- It simplifies:
  - refactoring/restructuring
  - maintenance
  - the introduction of new functionalities
- Test first help to build the documentation
  - Test cases are good “use samples”
  - Programming is more fun …

Example

- Existing system: a class current account to manage a bank account
  - deposit
  - withdraw
- Add a new functionality
  - settlement

Example:

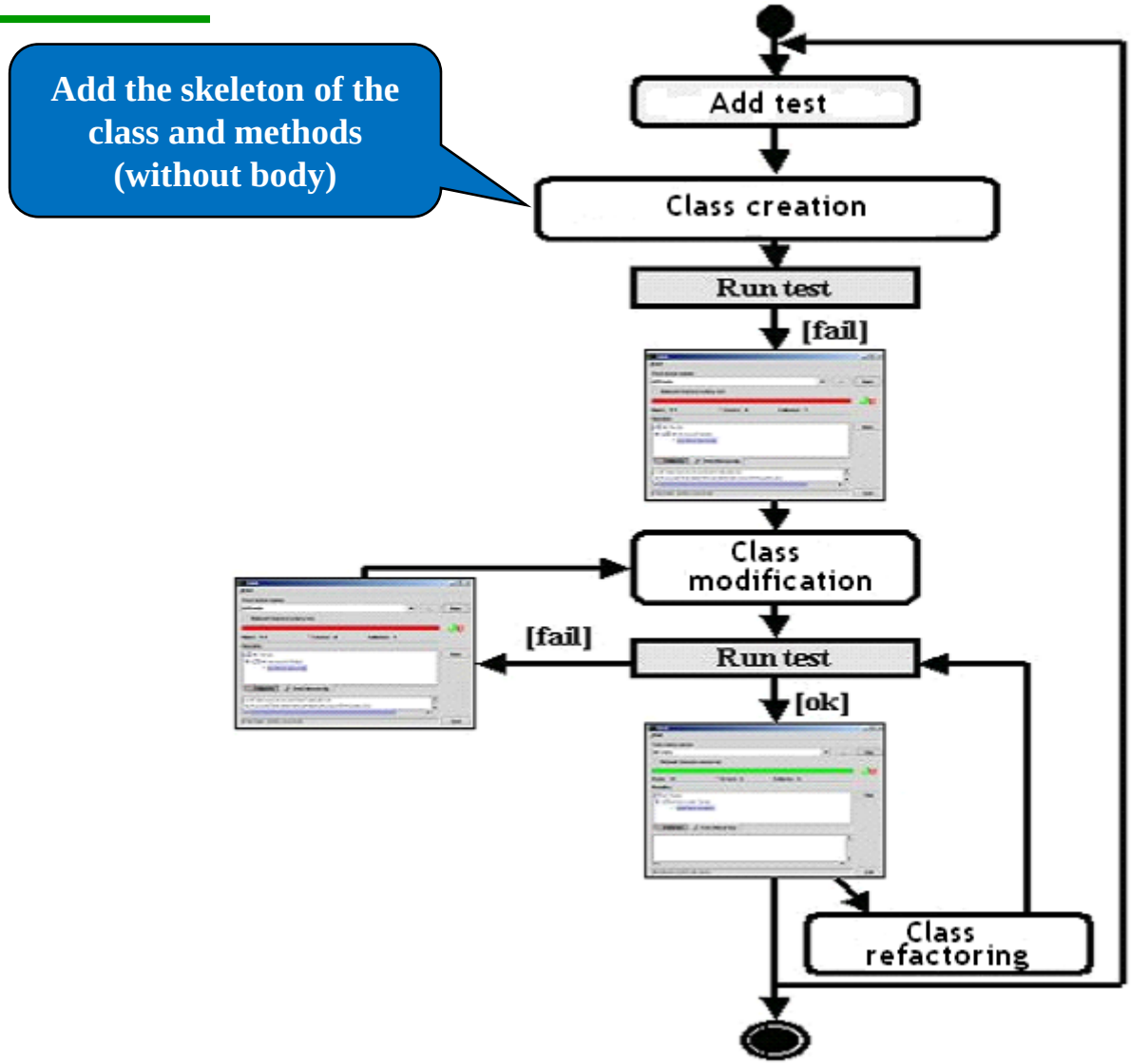
```
CurrentAccount cc = new CurrentAccount();
cc.deposit(12);
cc.draw(-8);
cc.deposit(10);
cc.settlement()
expected value: 14
```


Add a testcase
Add test
Class creation
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring

Add test cases for settlement

Create a test case

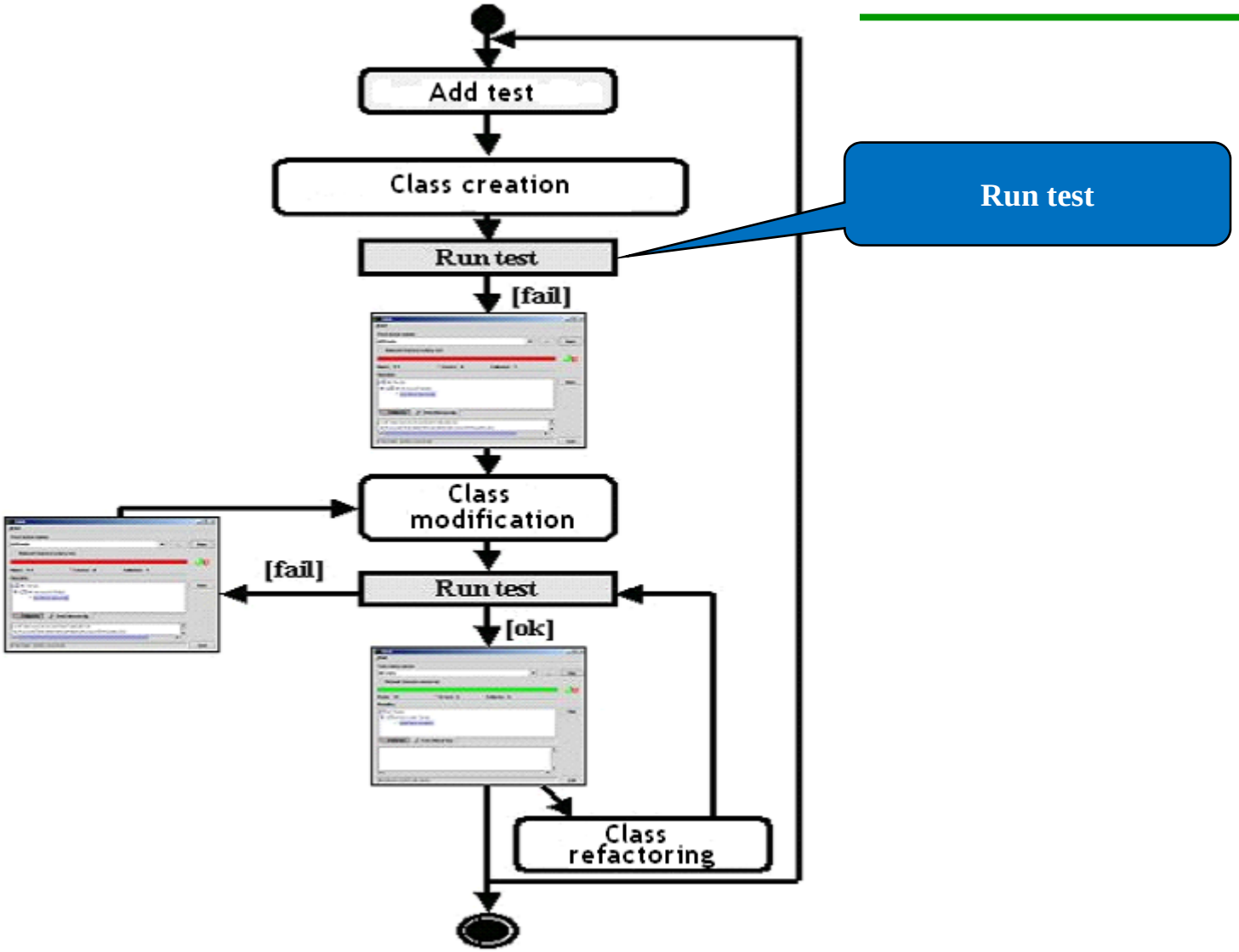
```
class Test_CurrentAccount extends TestCase{

 public void test_settlementVoid() {
    CurrentAccount c = new CurrentAccount();
    assertEquals(0, c.settlement());
 }

 public void test_settlement() {
    CurrentAccount c = new CurrentAccount();
    c.deposit(12);
    c.draw(-8);
    c.deposit(10);
    assertEquals(14, c.settlement());
 }
}
```


Add the skeleton of the class and methods (without body)
Add test
Class creation
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring

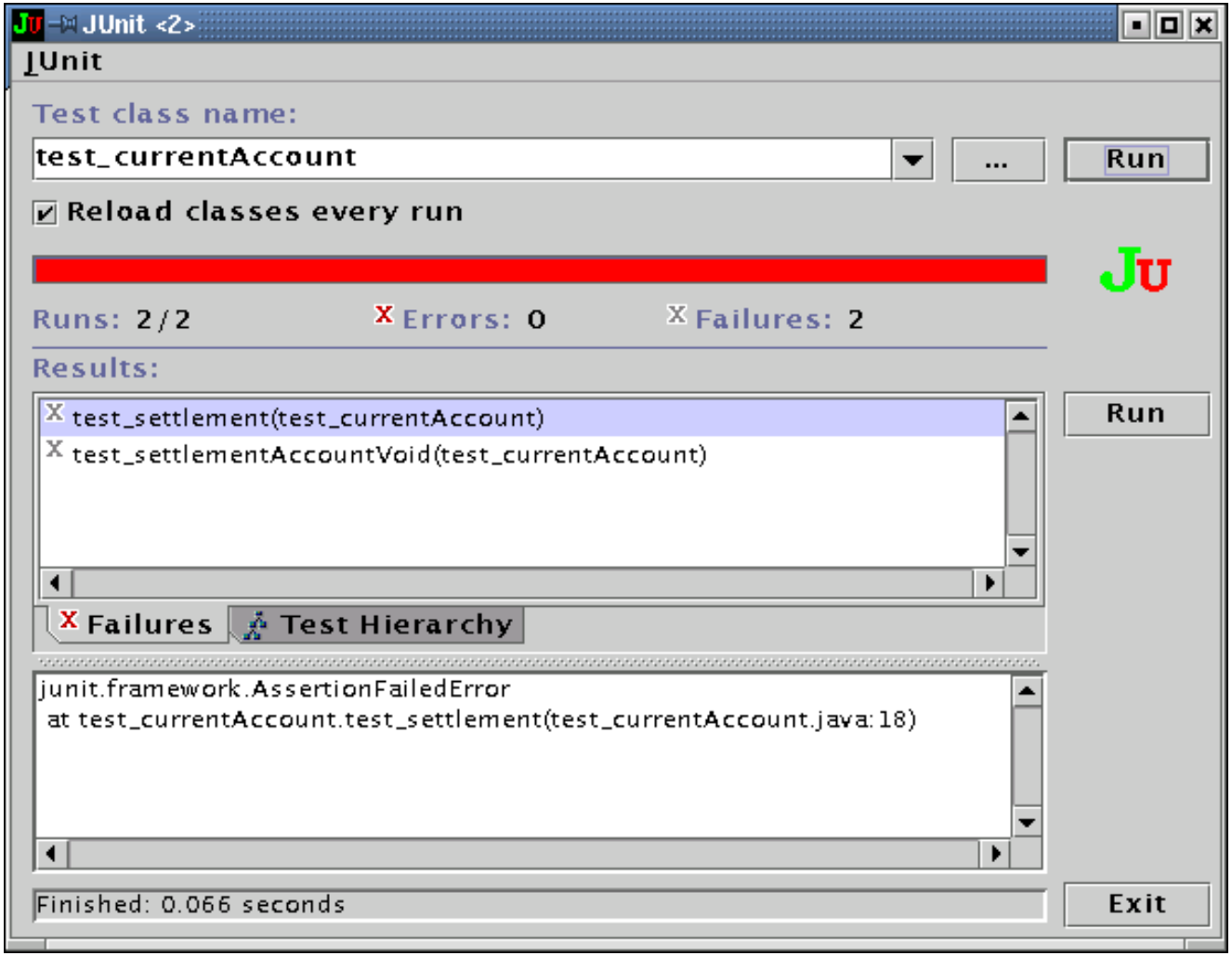
Add the Skeleton Code

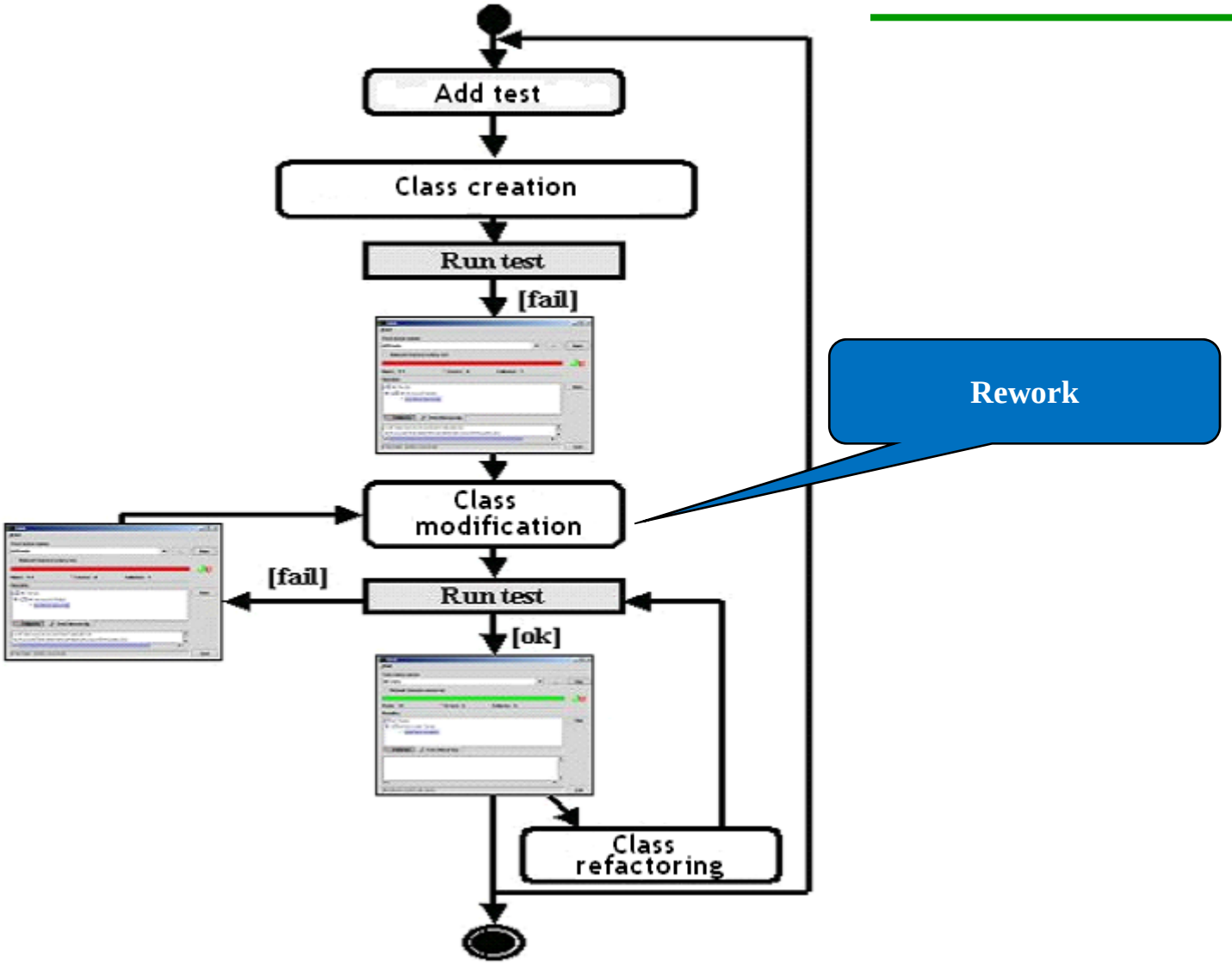
```
class CurrentAccount {
 int account[];
 int lastMove;

 CurrentAccount(){
  lastMove=0;
  account=new int[10];
 }

 public void deposit(int value){
  account[lastMove]=value;
  lastMove++;
 }

 public void draw(int value) {
  account[lastMove]=value;
  lastMove++;
 }

 public int settlement() {return -1;}

 public static void main(String args[]) {}
}
```

```
class Test_CurrentAccount extends TestCase{

 public void test_settlementVoid() {
    CurrentAccount c = new CurrentAccount();
    assertEquals(0, c.settlement());
 }

 public void test_settlement() {
    CurrentAccount c = new CurrentAccount();
    c.deposit(12);
    c.draw(-8);
    c.deposit(10);
    assertEquals(14, c.settlement());
 }
}
```


Add test
Class creation
Run test
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring

Run JUnit (first time)

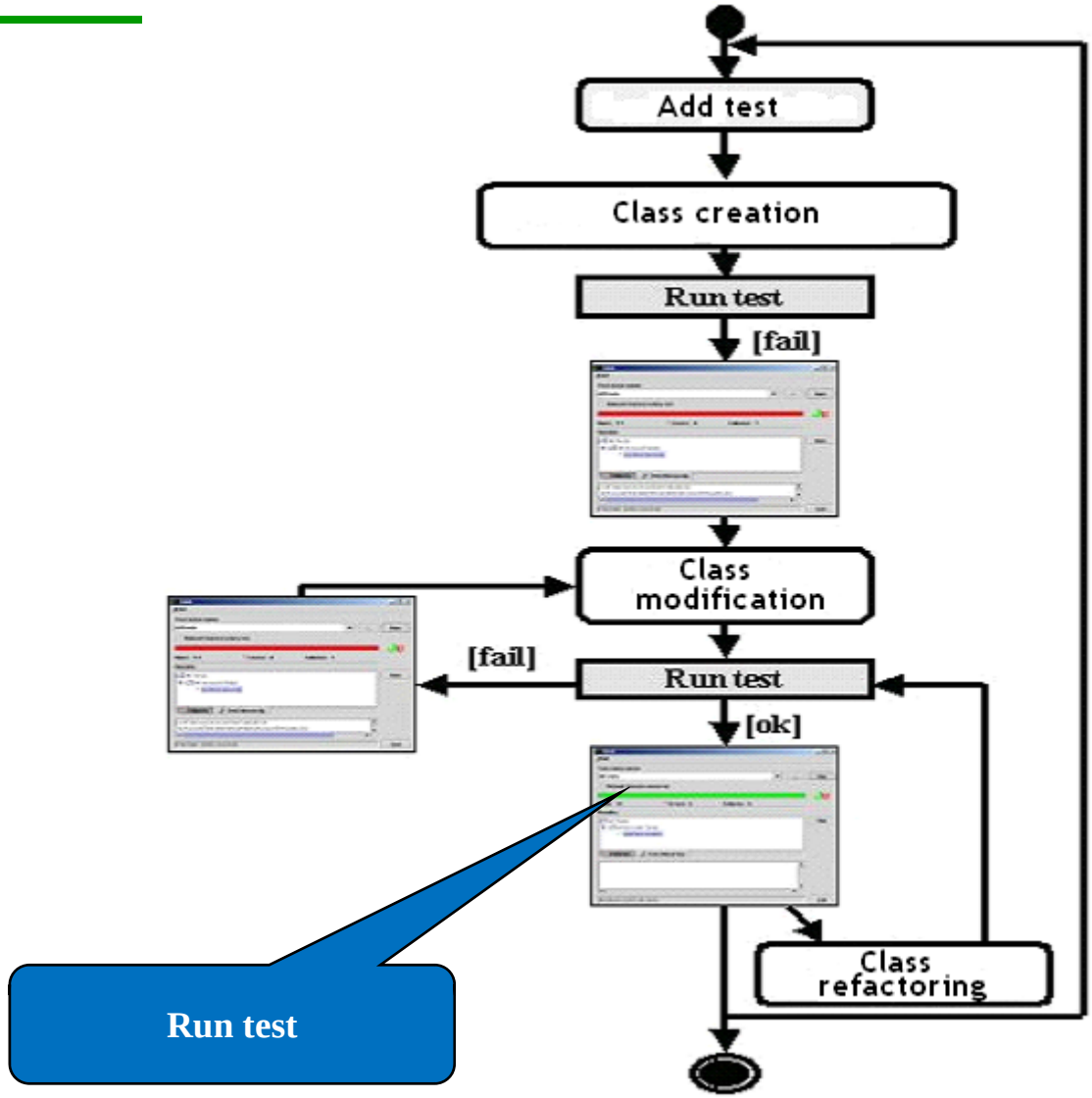



Add test
Class creation
Run test
[fail]
Rework
Class modification
[fail]
Run test
[ok]
Class refactoring

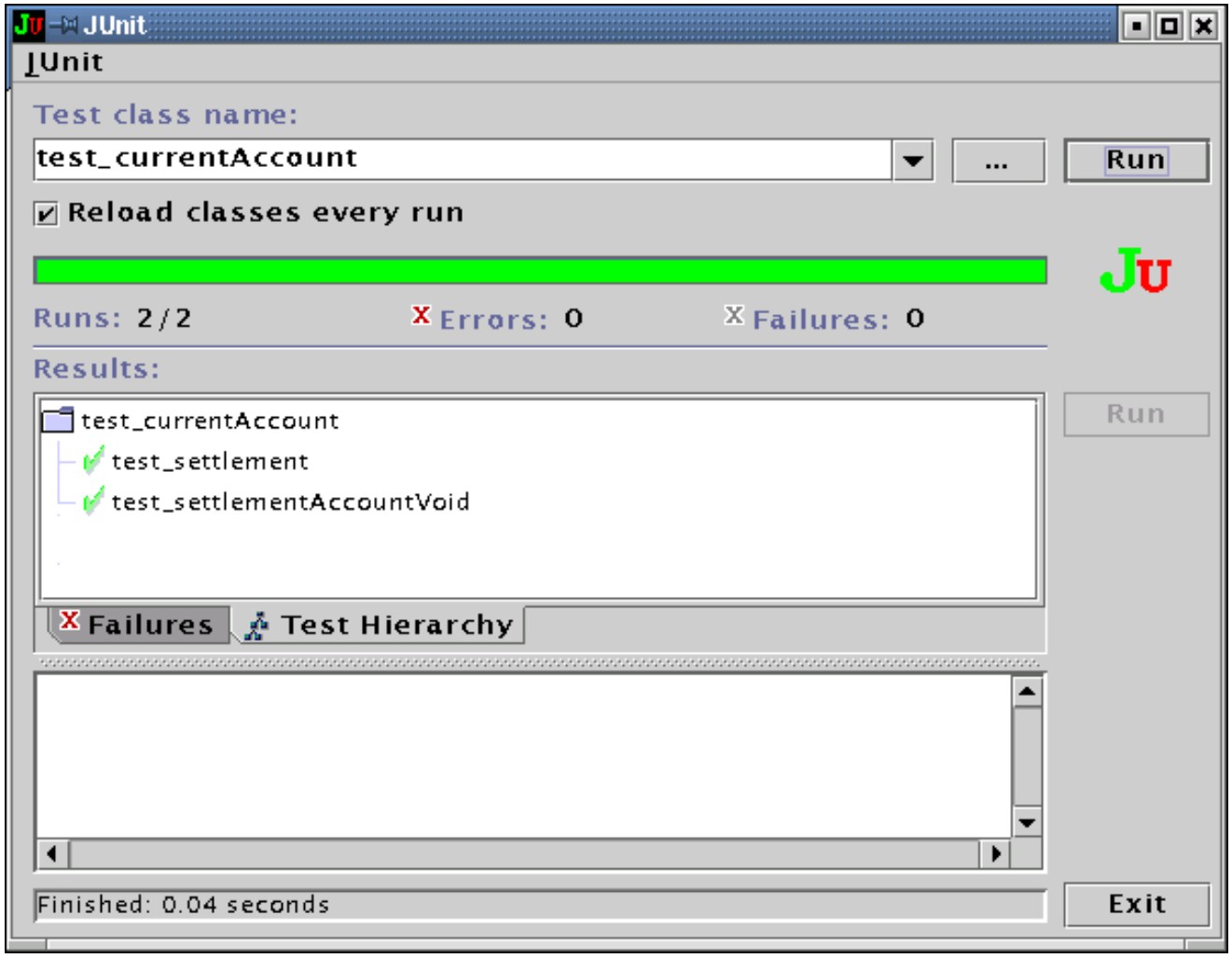
Rework

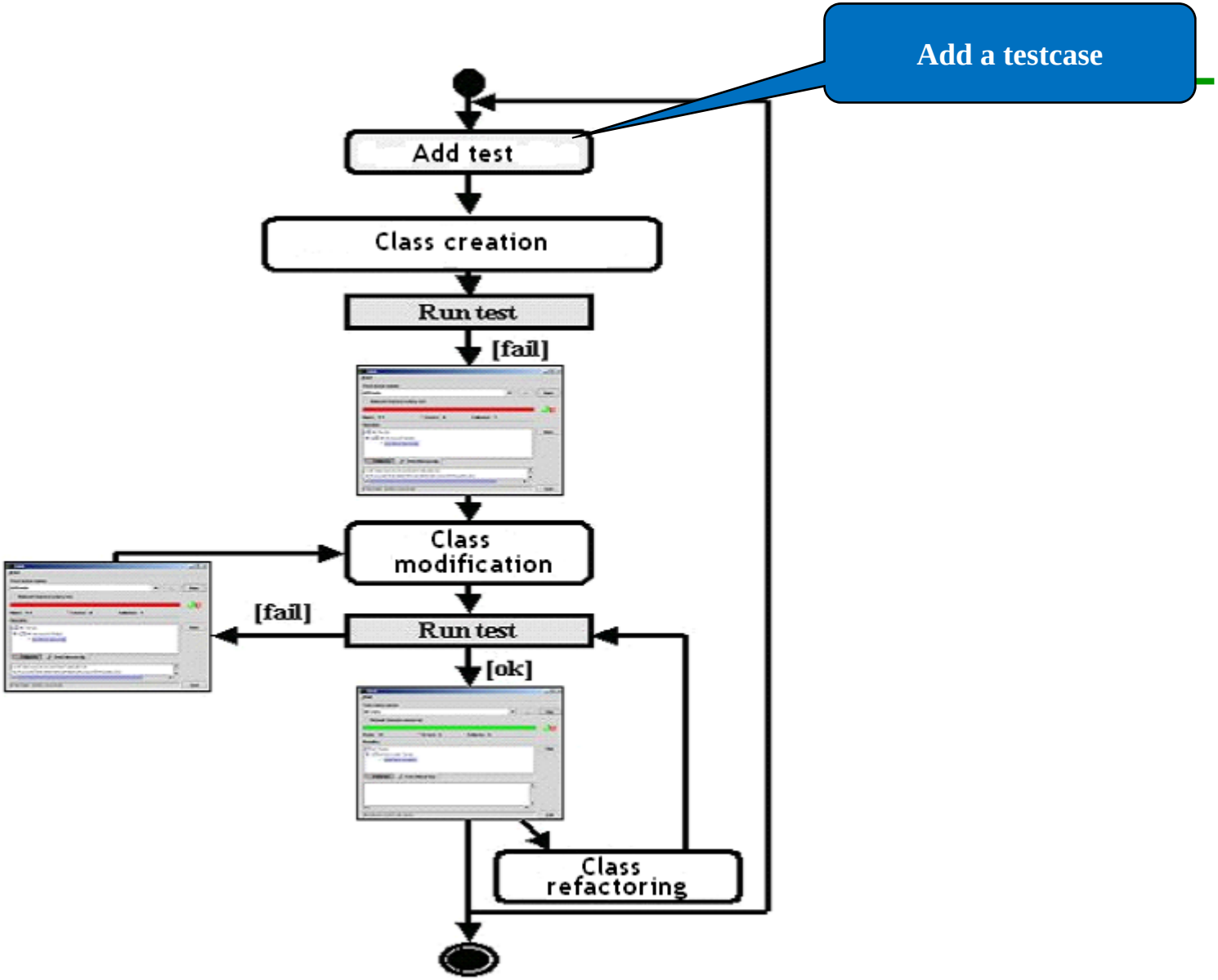
```
class CurrentAccount {
 int account[];
 int lastMove;

 CurrentAccount(){
  lastMove=0; account=new int[10];
 }

 public void deposit(int value){ …}

 public void draw(int value) { …}

 public int settlement() {
  int result = 0
  for (int i=0; i<account.length; i++) {
    result = result + account[i];
  }
  return result;
 }

 public static void main(String args[]) {}
}
```

```
class Test_CurrentAccount extends TestCase{

 public void test_settlementVoid() {
    CurrentAccount c = new CurrentAccount();
    assertEquals(0, c.settlement());
 }

 public void test_settlement() {
    CurrentAccount c = new CurrentAccount();
    c.deposit(12);
    c.draw(-8);
    c.deposit(10);
    assertEquals(14, c.settlement());
 }
}
```


Add test
Class creation
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring
Run test

Run JUnit (second time)

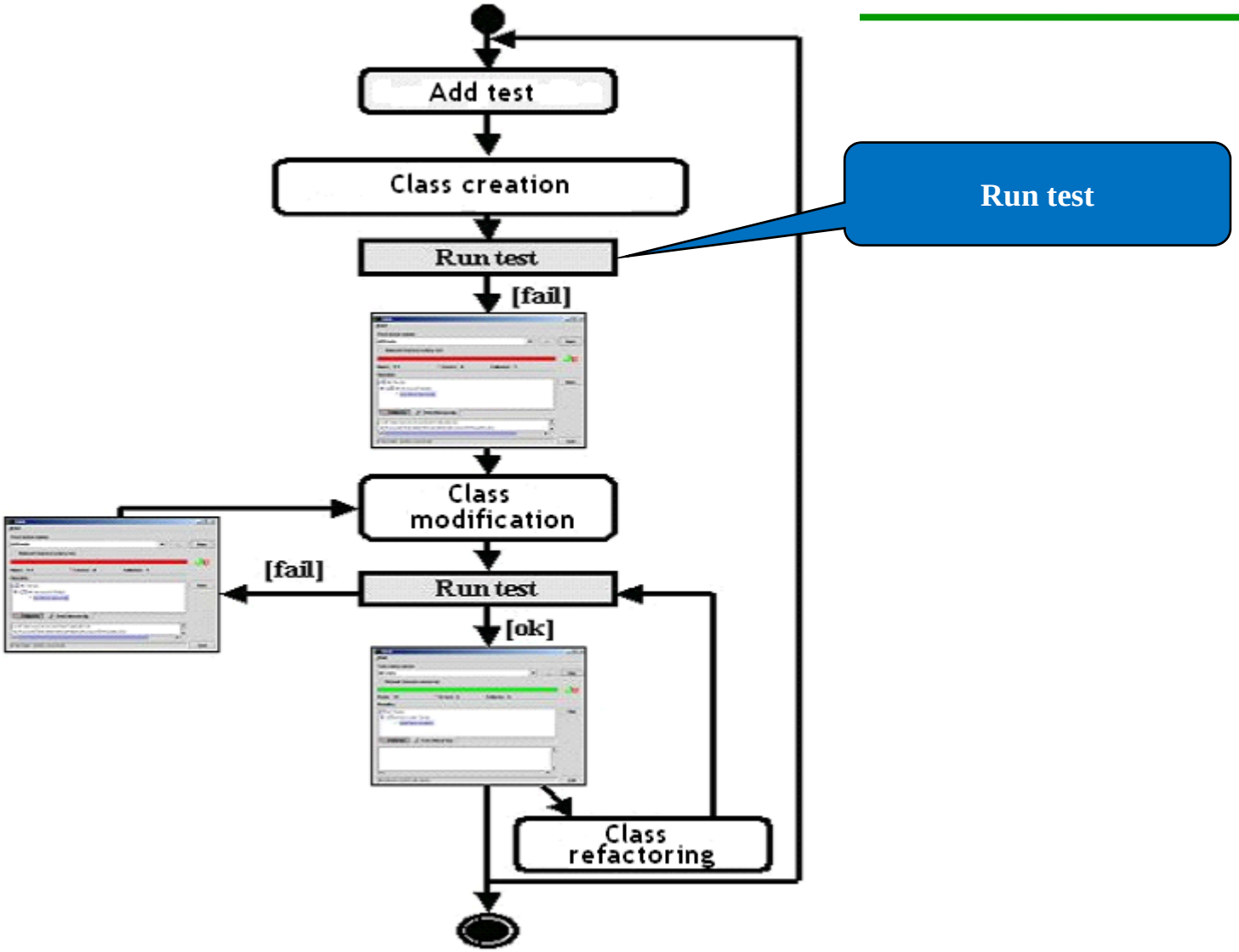



Add a testcase
Add test
Class creation
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring

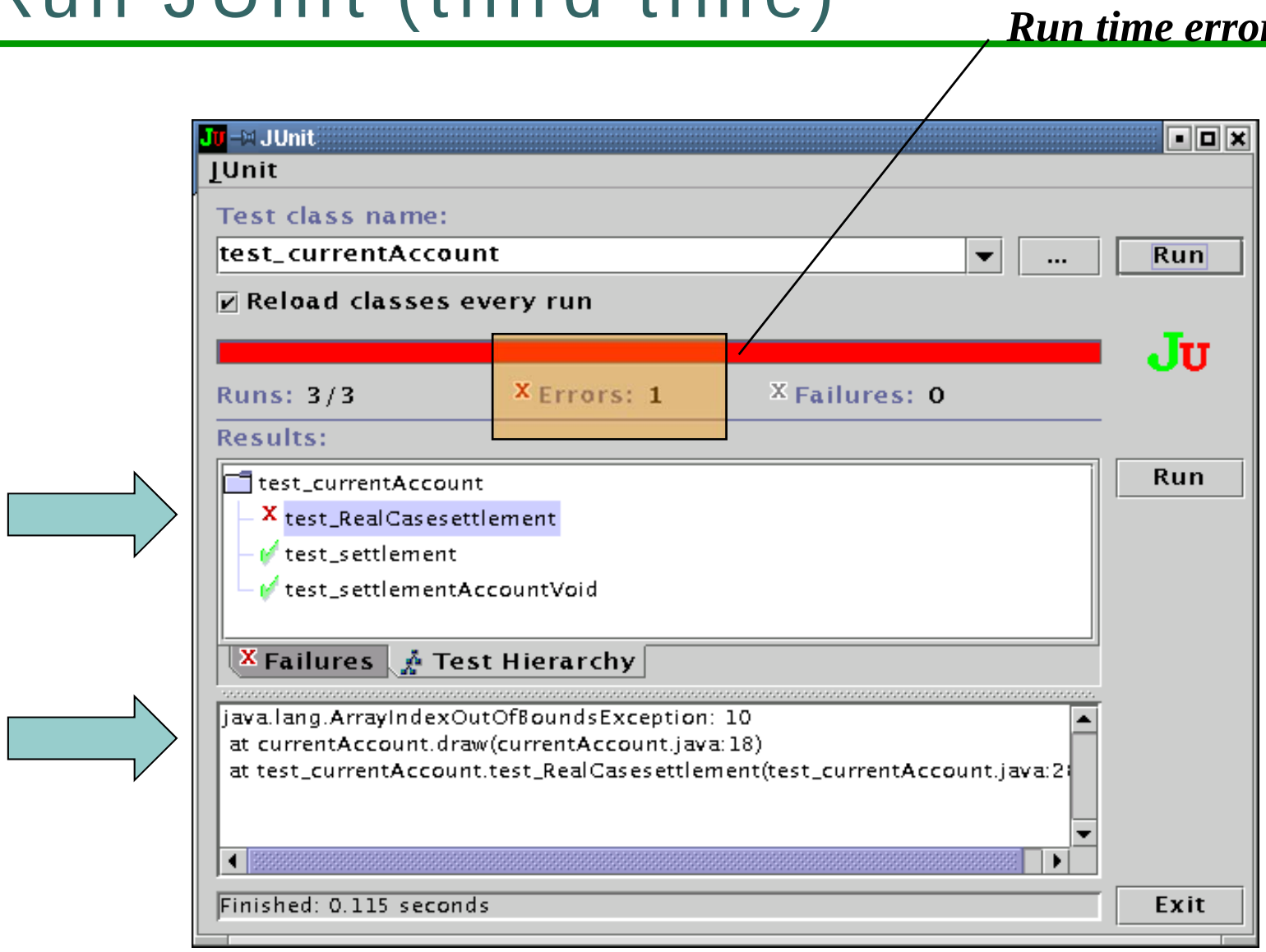
Add a New Test Case

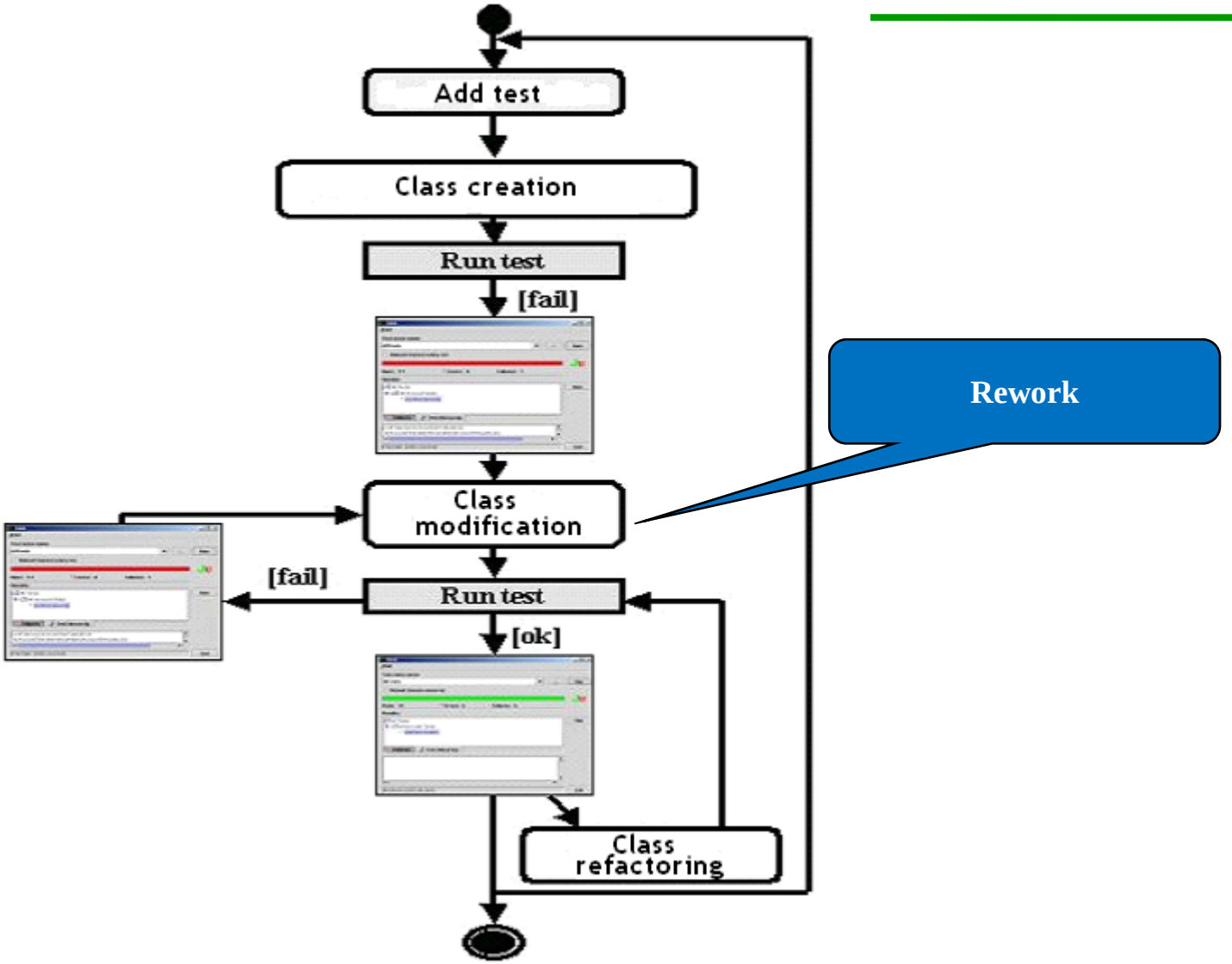
```
class CurrentAccount {
 int account[];
 int lastMove;

 CurrentAccount(){
  lastMove=0; account=new int[10];
 }

 public void deposit(int value){ …}

 public void draw(int value) { …}

 public int settlement() {
  int result = 0;
  for (int i=0; i<account.length; i++) {
     result = result + account[i];
  }
  return result;
 }

 public static void main(String args[]) {}
}
```

```
class Test_currentAccount extends TestCase{

…

public void test_realCaseSettlement() {
  currentAccount c = new currentAccount();
  for (int i=0; i <10 ; i++)
         c.deposit(1);
  c.draw(-10);
  assertEquals(0, c.settlement());
 }

}
```


Add test
Class creation
Run test
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring

Run JUnit (third time)

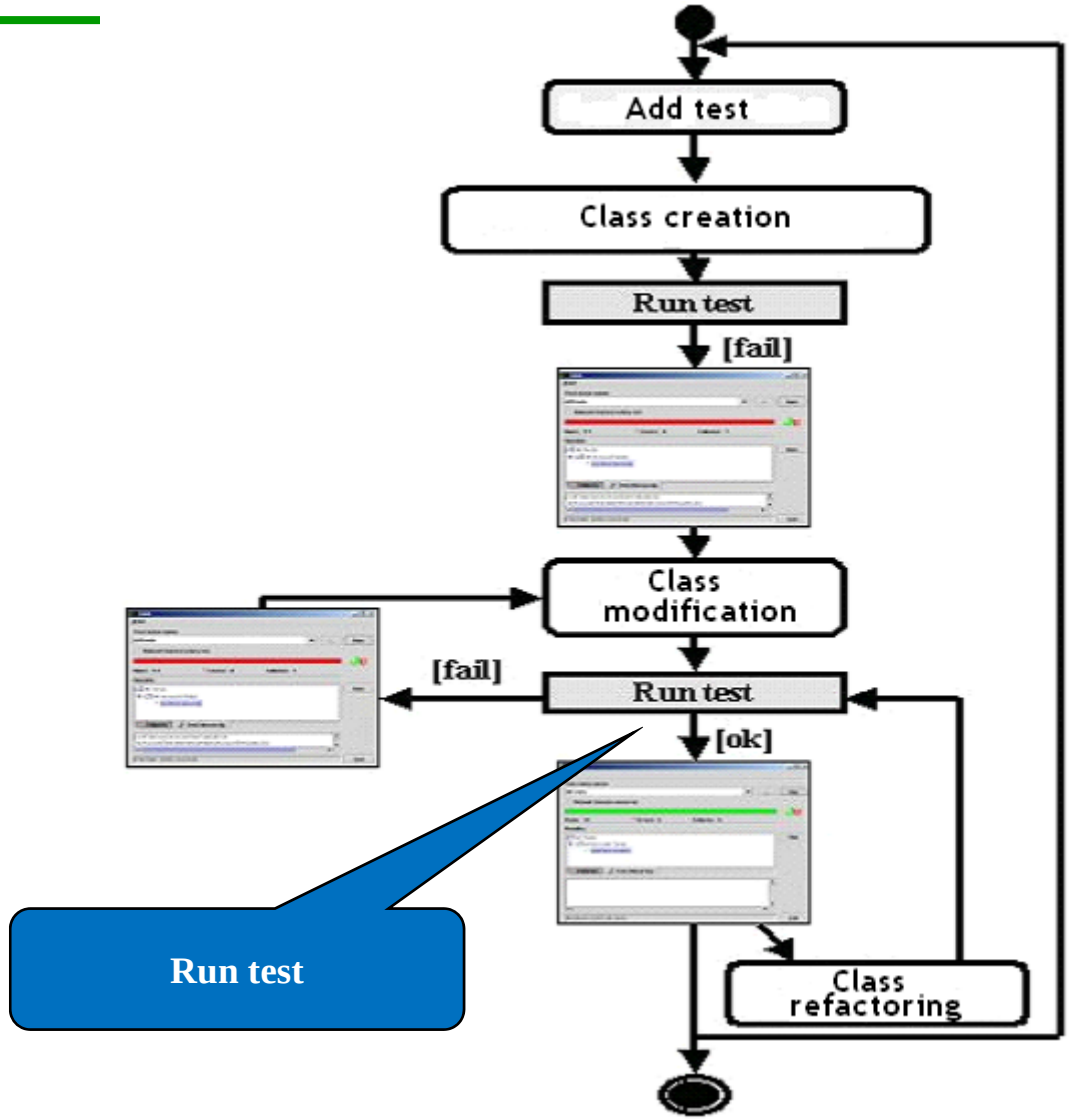



Add test
Class creation
Run test
[fail]
Rework
Class modification
[fail]
Run test
[ok]
Class refactoring

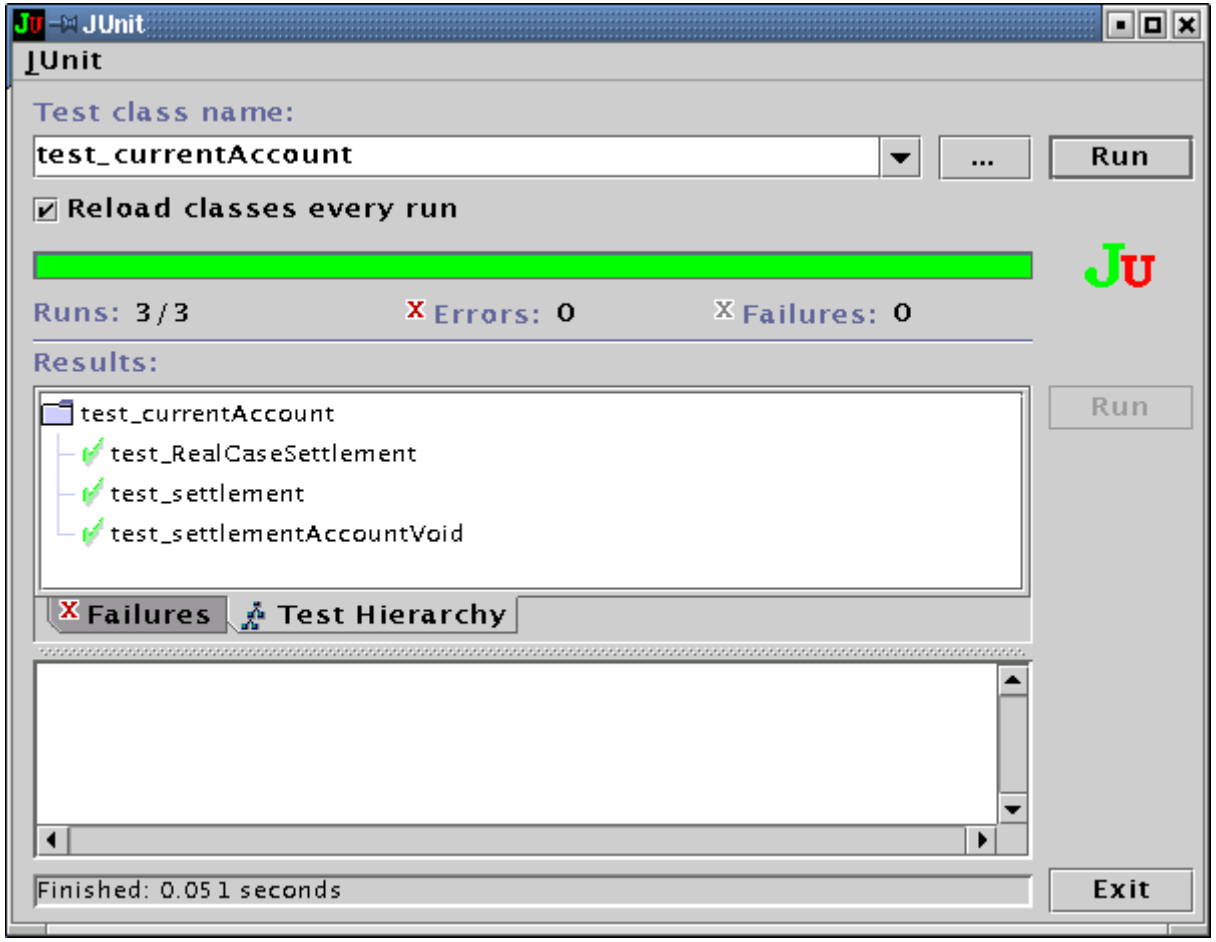
Rework

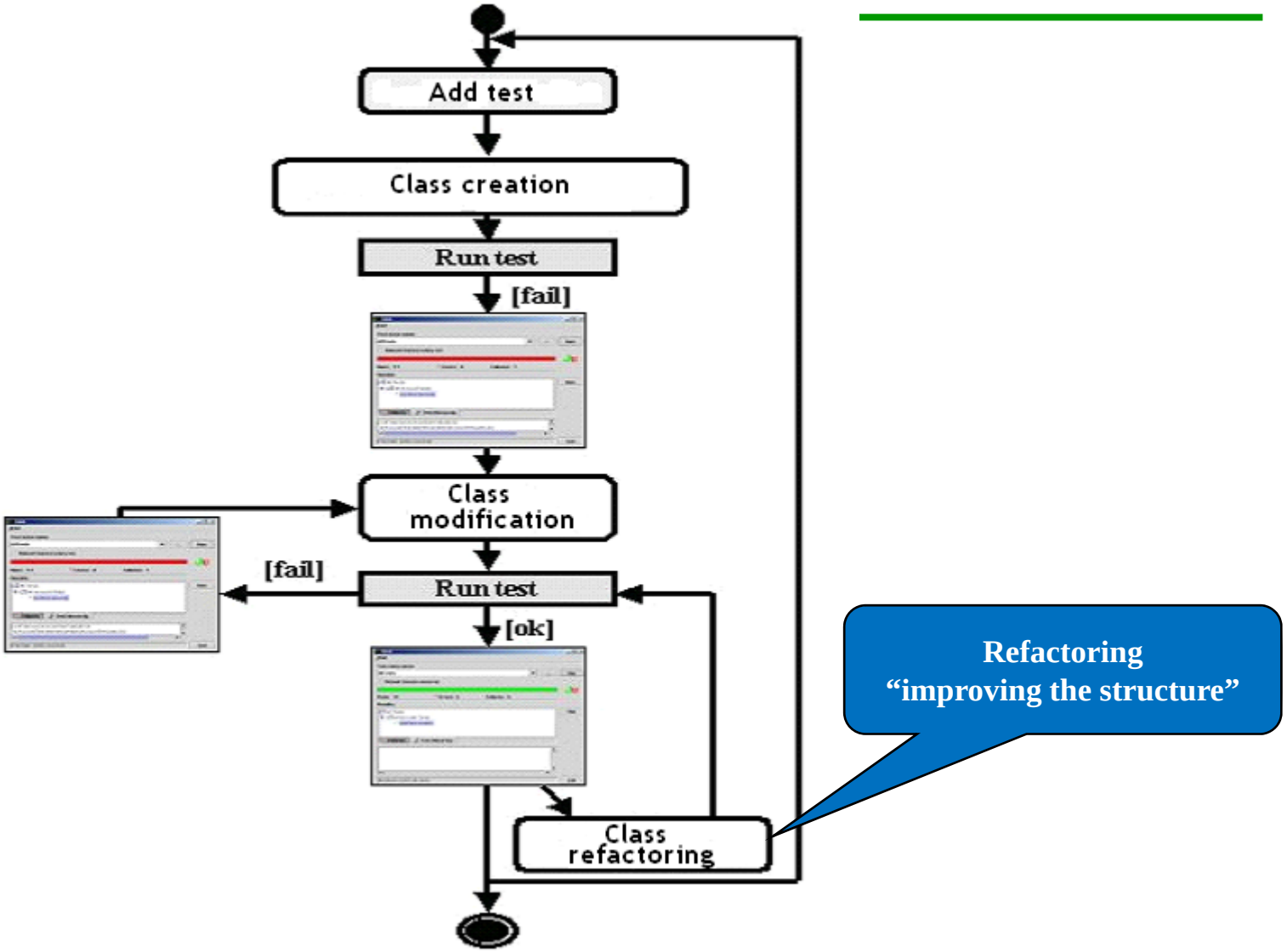
```
class CurrentAccount {
 int account[];
 int lastMove;

 CurrentAccount(){
  lastMove=0; account=new int[100];
 }

 public void deposit(int value){ …}

 public void draw(int value) { …}

 public int settlement() {
  int result = 0
  for (int i=0; i<account.length; i++) {
     result = result + account[i];
  }
  return result;
 }

 public static void main(String args[]) {}
}
```

```
class Test_currentAccount extends TestCase{

…

public void test_realCaseSettlement() {
  currentAccount c = new currentAccount();
  for (int i=0; i <10 ; i++)
         c.deposit(1);
  c.draw(-10);
  assertEquals(0, c.settlement());
 }

}
```


Add test
Class creation
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring
Run test

Run JUnit (fourth time)

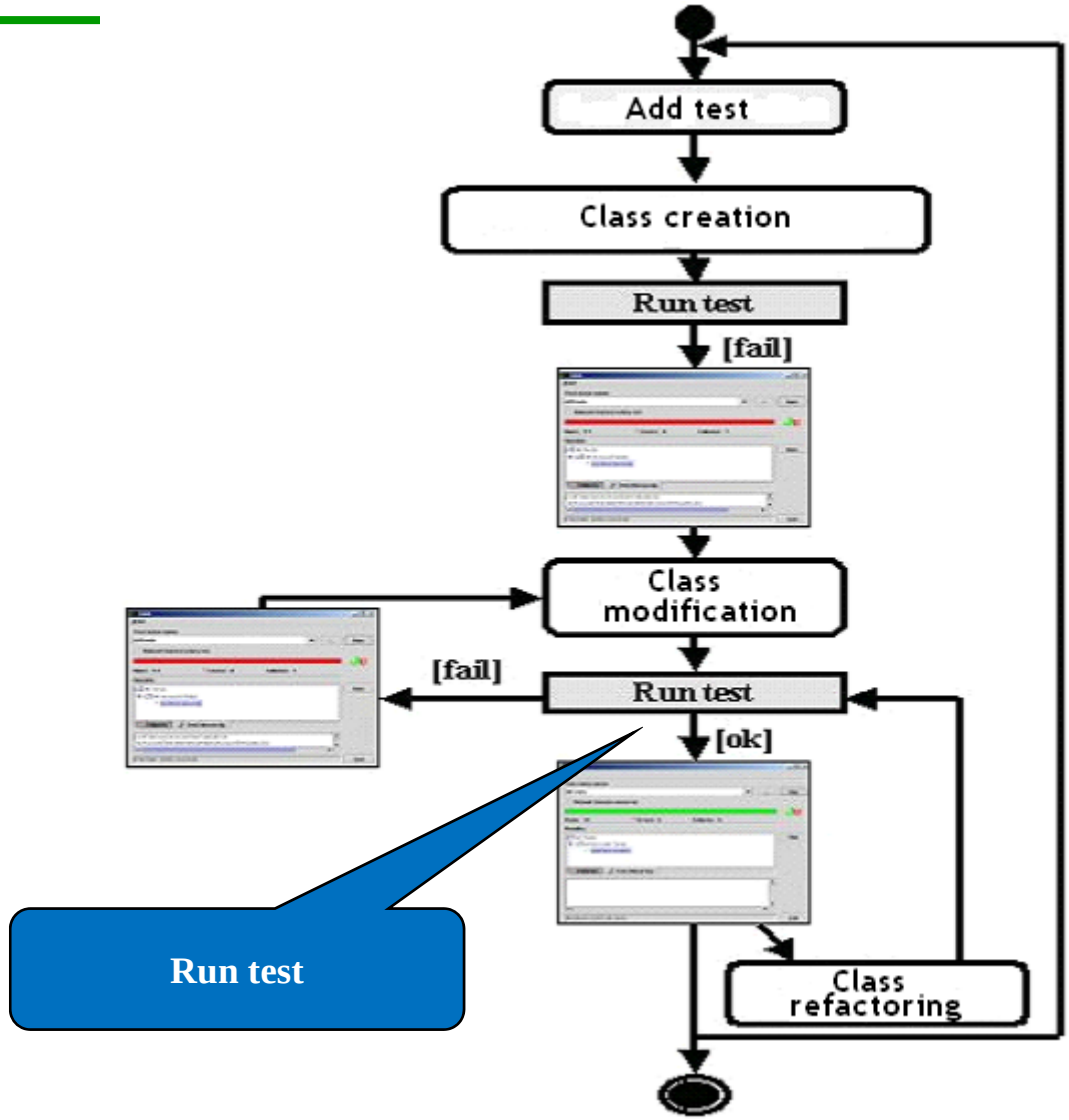



Add test
Class creation
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring
Refactoring
“improving the structure”

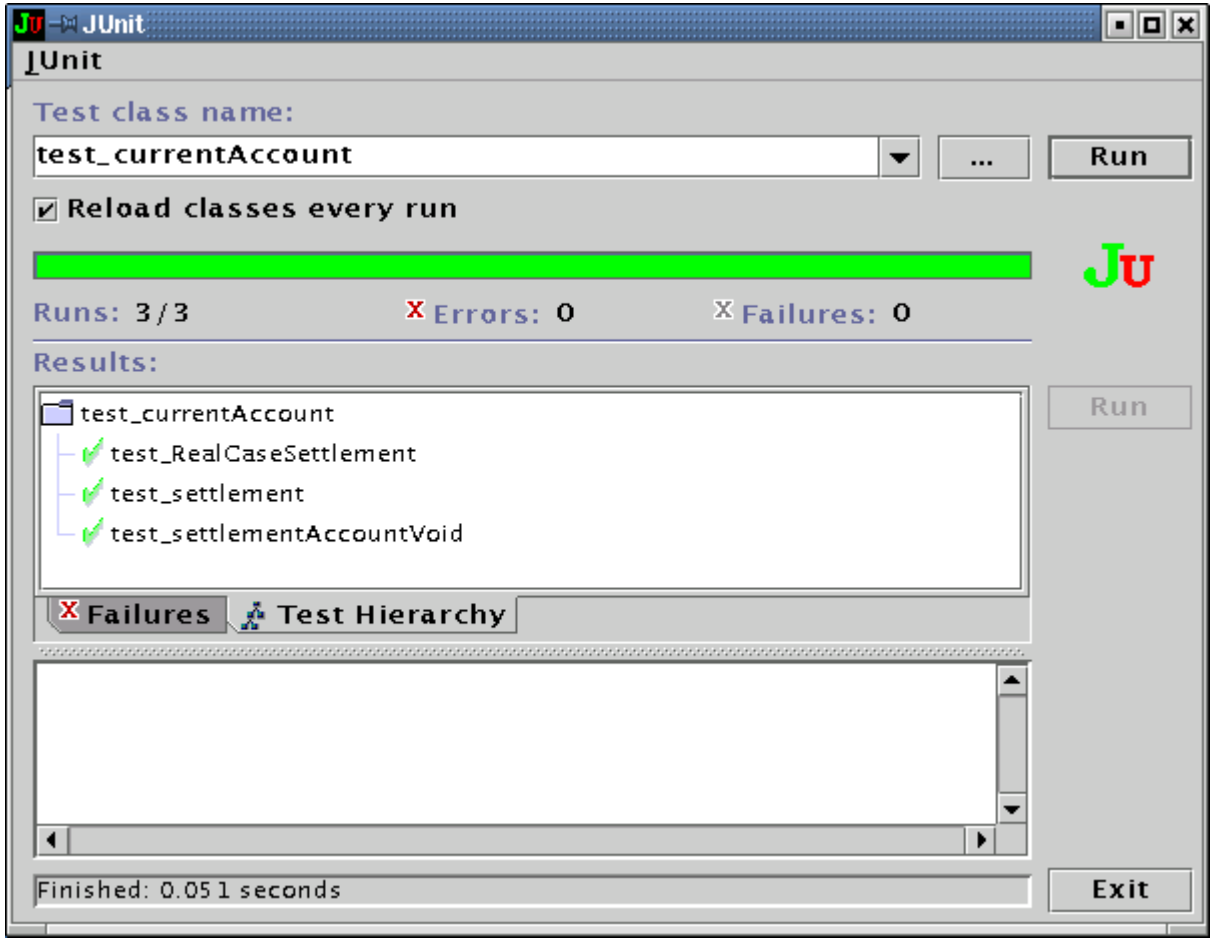
Refactoring

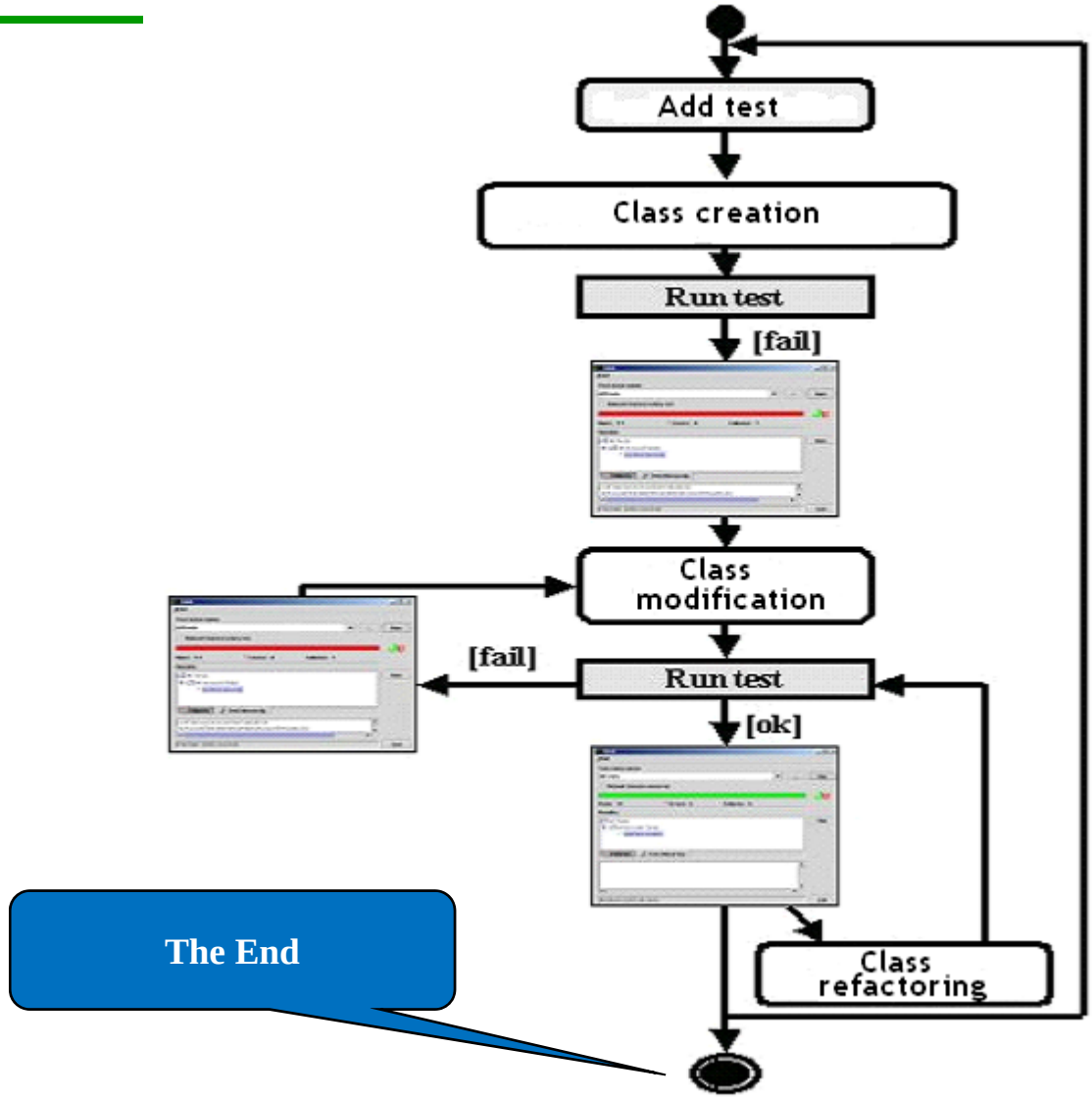
```
public class CurrentAccount {
        List account = new LinkedList();

        public void deposit(int value) {
                account.add(new Integer(value));
        }

        public void draw(int value) {
                account.add(new Integer(value));
        }

        public int settlement() {
                int result = 0;
                Iterator it=account.iterator();
                while (it.hasNext()) {
                        Integer value_integer = (Integer)it.next();
                        int val = value_integer.intValue();
                        result = result + val;
                }
                return result;
        }
}
```

"changing the data structure: Array --> List"


Add test
Class creation
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring
Run test

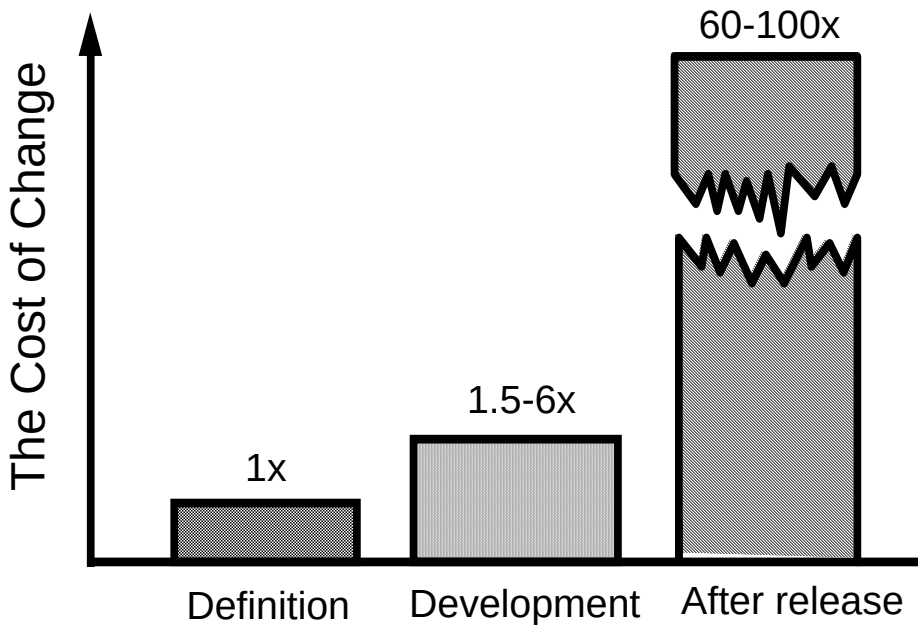
Run JUnit (fifth time)




Add test
Class creation
Run test
[fail]
Class modification
[fail]
Run test
[ok]
Class refactoring
The End

Review: Life-Cycle Models

- Code-and-fix
- Waterfall
- Spiral
- Agile



- Which one make testing easier?
  - The ease of plan & design
  - The ease of fixing bugs

Understanding Test Management

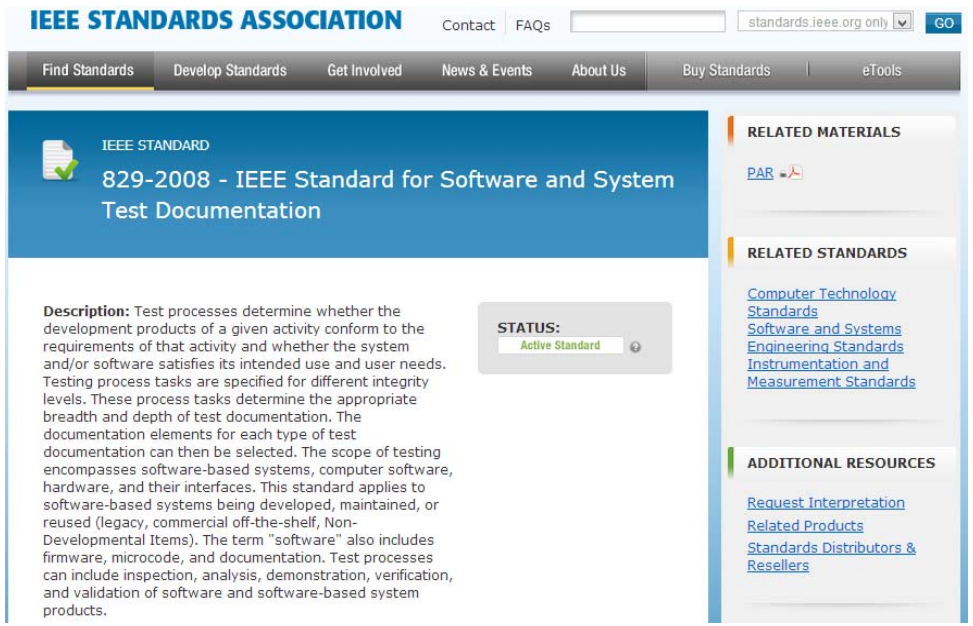
- Test Process Overview

- IEEE 829 Software Test Documentation Standard

IEEE 829 Standard

- IEEE Standard for Software and System Test Documentation (old version: 829-1998, new version: 829-2008)
  - Directly cover the documentation aspect.
  - Indirectly cover the process/schedule aspect.

http://standards.ieee.org/findstds/standard/829-2008.html



Benefits:

- Something to base your thinking for an inexperienced testing team.
- Well written example uses of the templates.
- Well-recongized standard

Drawbacks:

- Not so easy to digest, required many readthroughs to find out for what a particular document is really used
- Too many documents, we had to modify and merge some of them

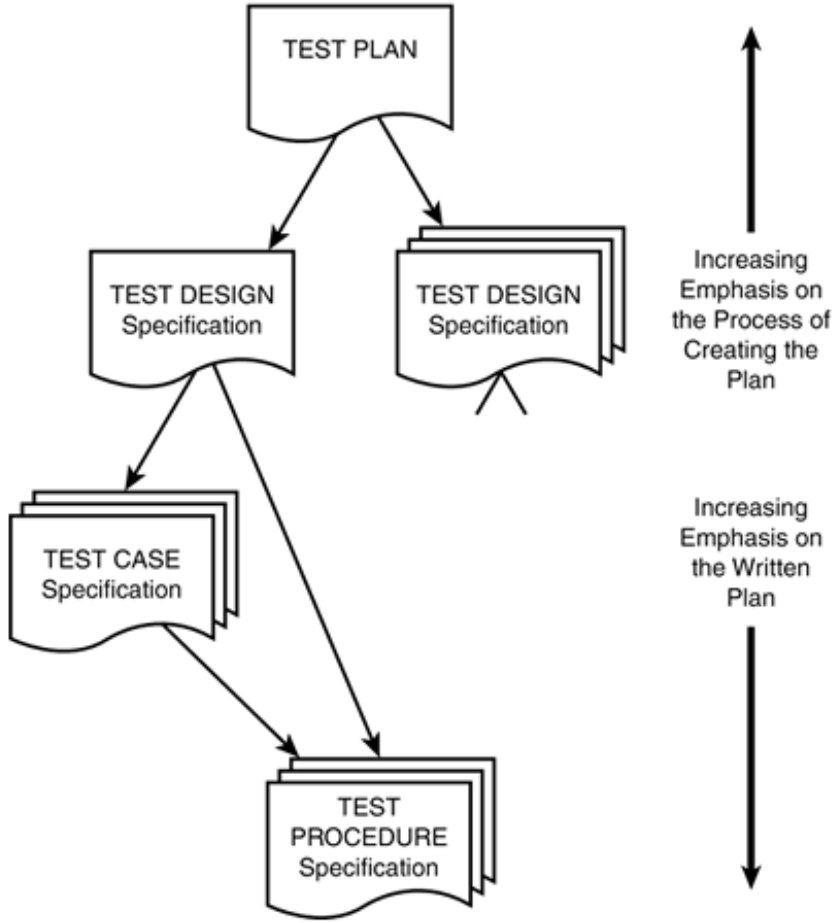
Overview of IEEE 829-1998

- Describe format and content for the following documents:

Test Plan (测试计划)	Plan
Test Design Specification (测试设计规范) **Test Case Specification** (测试用例规范) **Test Procedure Specification** (测试过程规范) • Supplemented by test scripts if test automation is applied.	Design
Test Log (测试记录) **Test Incident Report** (测试错误报告) • Also know as *bug report* or *anomaly report* **Test Summary Report** (测试摘要报告) • Or *test report* for short.	Execution

Their Relation

Figure 18.1. The different levels of test documents all interact and vary on whether their importance is the document itself or the process of creating it.

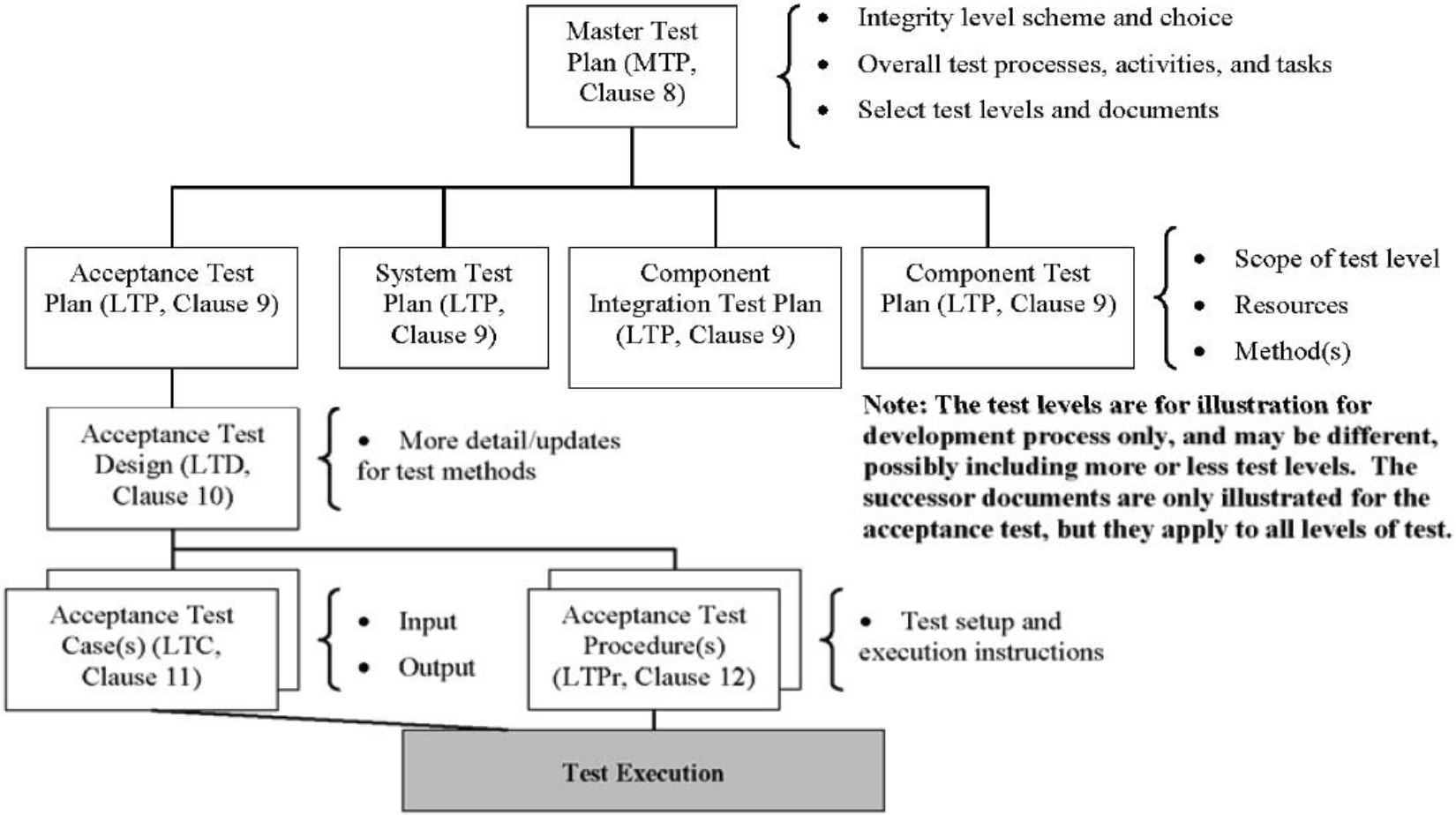


New in IEEE 829-2008

- Moves away from stand-alone documents to structural documents.
  - Eliminate redundancy of information in various test documents (e.g. unit test plan and system test plan can share certain content)
- Add new document types
  - MTP and LTP
    - ***Master Test Plan*** (MTP, 主测试计划) governs the management of the overall test effort.
    - ***Level Test Plans*** (LTP，子测试计划) covers particular levels of testing (unit/component level, integration level, system levels, acceptance levels)
  - MTR and LTR
    - ***Master Test Report*** (MTR，主测试报告) consolidates LTRs and summarizes the results of the tasks identified in the Master Test Plan.
    - ***Level Test Reports*** (LTR，子测试报告) summarizes the results of particular levels of testing.
  - ***Level Interim Test Status Report***（测试进度中期报告）
    - Report test progress of particular levels of testing

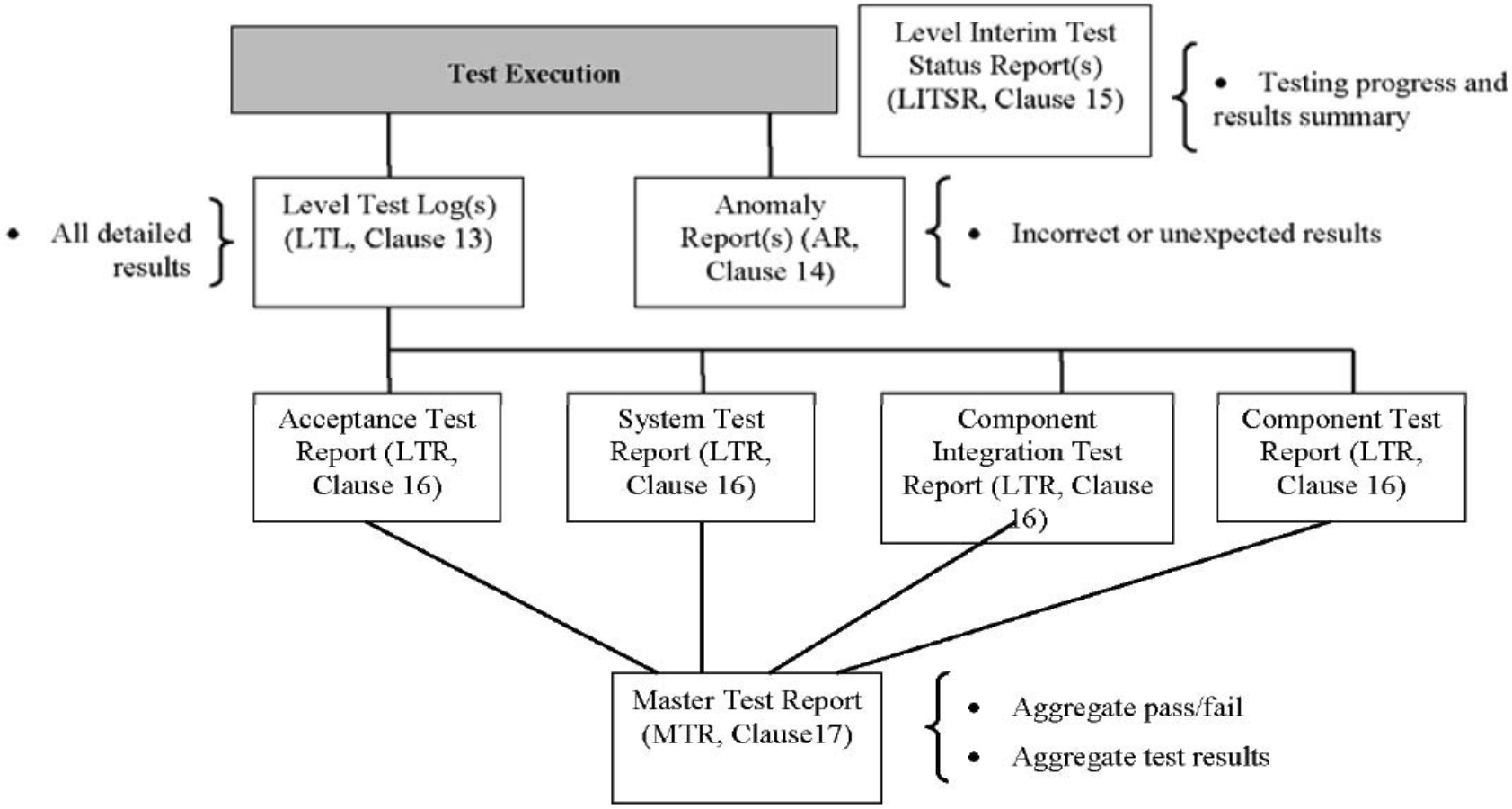
New in IEEE 829-2008

- Document structure as prescribed in 829-2008



New in IEEE 829-2008

- Document structure as prescribed in 829-2008



New in IEEE 829-2008

- Introduce the concepts of **integrity levels**
  - Four levels: catastrophic, critical, marginal, negligible.
    - Describe the importance of the software aspects to the user.
    - The process of identifying the integrity level is the **criticality analysis**.
  - Define recommended minimum testing tasks for each integrity level.
- Detailed criteria for testing tasks
  - Defines specific criteria for each testing task including minimum recommended criteria for correctness, consistency, completeness, accuracy, readability, and testability.

Understanding IEEE 829

- Test Planning（如何做测试计划）
  - IEEE 829 Document: Test Plan
- Test Design （如何做测试设计）
  - IEEE 829 Documents: Test Design Specification, Test Case Specification, Test Procedure Specification
  - Tool: TestLink
- Test Report（如何编写测试报告）
  - IEEE 829 Documents: Test Log, Test Incident Report, Test Summary Report
  - Tool: Bugzilla

Understanding IEEE 829

- Test Planning（如何做测试计划）
  - IEEE 829 Document: Test Plan
- Test Design （如何做测试设计）
  - IEEE 829 Documents: Test Design Specification, Test Case Specification, Test Procedure Specification
  - Tool: TestLink
- Test Report（如何编写测试报告）
  - IEEE 829 Documents: Test Log, Test Incident Report, Test Summary Report
  - Tool: Bugzilla

Overview

- **The software test plan** is the primary means by which software testers communicate to the product development team what they intend to do.
- **IEEE Standard 829-1998** states that the purpose of a software test plan is as follows:
  - To prescribe the scope, approach, resources, and schedule of the testing activities. To identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with the plan.
- Provides a feasibility check of
  - Resources/Cost
  - Schedule
  - Goal

IEEE 829-1998: Test Plan

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Features to be Tested
6. Features not to be Tested
7. Approach
8. Item Pass/Fail Criteria
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Needs
13. Staffing and Training Needs
14. Responsibilities
15. Schedule
16. Planning Risks and Contingencies
17. Approvals
18. Glossary

We will use two case studies to facilitate our learning on this format

Case Studies

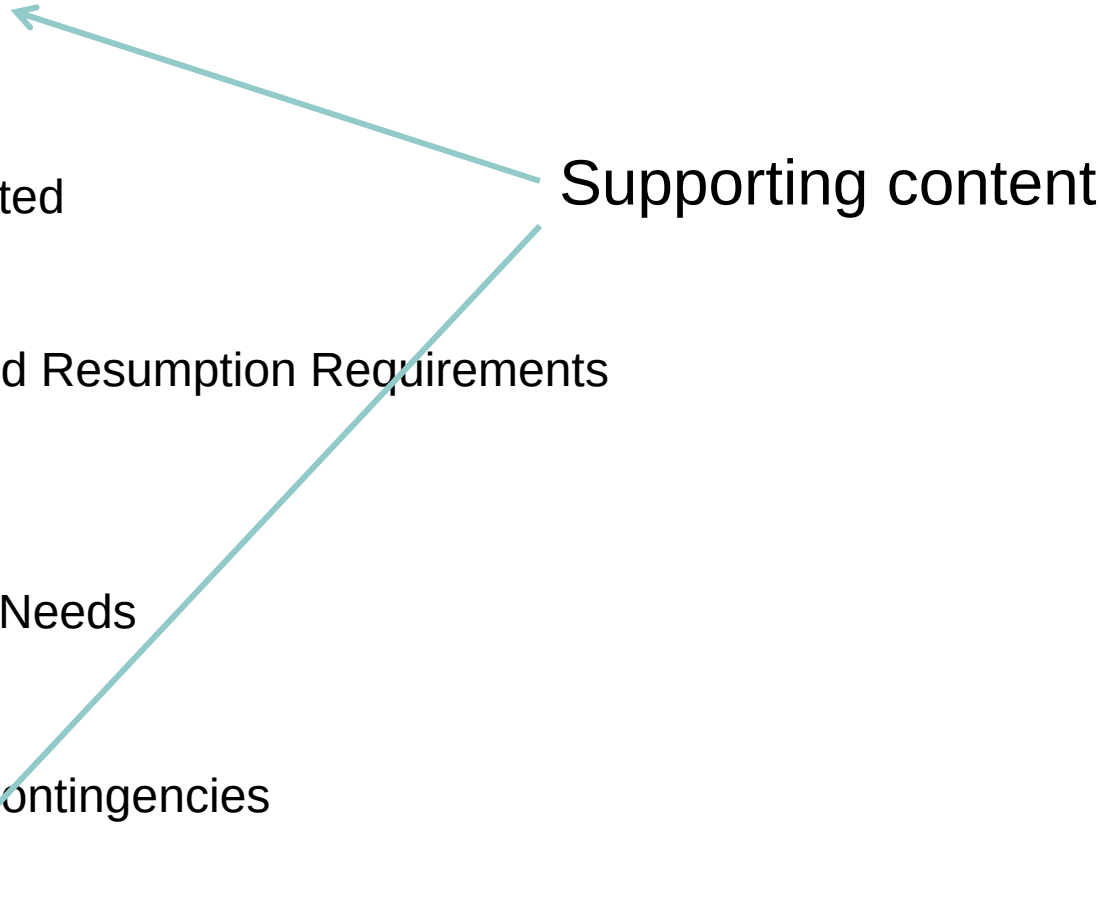
- **Case study 1**（智能交通）: Transport4You, an Intelligent Public Transportation Manager that offer personalized services to citizens. The features include:
  - allow a citizen to register into the system and pre-pay for a certain number of trips.
  - recognize when a registered citizen gets on a bus and determine the journey he/she performs, calculating the fare he/she has to pay, and detracting it from his/her credit;
  - when the citizen's credit is finished, allow him/her to pay the bus fare through the cell phone;
  - provide registered citizens with information about changes in the lines they use most frequently.
  - offer suggestions to registered citizens for alternative paths, for example because it determines that there are routes that are more optimized than the ones they usually take, or because there is a problem on some line that affects their intended path.

Case Studies

- **Case study 2**（人肉搜索）: MOb, the Mass Observation project support the study of a subject through in-situ observations by a collection of people. The features include:
  - Allows a person (the *Initiator*) to enter a question (the *observation event*) and to make it available to a set of people (the *observers*). The Initiator has the ability to choose the group by name/email, by region, or by other discernible attributes.
  - Allows observers to answer questions posed by the Initiator by making notes, taking pictures, providing audio commentary.
  - The system automatically collect observations and making them available for analysis and viewing. The Initiator may choose to make observations public with his own summary, or to simply use the data privately.
  - The system will run on mobile devices such as an Android phone.

IEEE 829-1998: Test Plan

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Features to be Tested
6. Features not to be Tested
7. Approach
8. Item Pass/Fail Criteria
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Needs
13. Staffing and Training Needs
14. Responsibilities
15. Schedule
16. Planning Risks and Contingencies
17. Approvals
18. Glossary



Supporting Content

- Header
  - Document identifier
  - Current version number
  - Date of writing
  - Revision history of the document
- Introduction
  - Relevance and possible uses of the document
  - The objective of testing
  - Intended audience: who should read it, and why?
  - Reference to other documents and artifacts needed for understanding this document
- Appendix
  - Approval: persons responsible for the document
  - Glossary: technical terms used in the document

On the Objective(s) of Testing

- Generic : “Meets customer requirements or satisfaction” is a bit too broad.
- Better examples commonly found in test plan:
  - Ensure that all the (requirements) features exist in the system
  - Ensure that the developed components and the purchased components integrate as specified
  - Ensure that performance targets (response time, resource capacity, transaction rate, etc.) are met.
  - Ensure the system is robust (stress the system beyond boundary)
  - Ensure that the user interface is “clear,” “novel,” or “catchy,” etc.
  - Ensure that the required functionality and features are “high quality.”
  - Ensure that industry standards are met.
  - Ensure that internationalism works (laws and looks)
  - Ensure that the software is migratable

On the Objective(s) of Testing

- Consider each quality factor:

Functionality	Availability
	Correctness
	Compliance
Efficiency	Performance testing
	Load testing
	Stress testing
Security	Privacy
	Integrity
Usability	Understandability
	Learnability
	Operability
	Likeability
Reliability	Availability
	Fault tolerance
	Recoverability
Portability	Adaptability
	Installability
	Conformance
	Co-existence

On the Approval

- Should involve other relevant departments, such as the development team members.
- This approach allows the author to hear other voices on the test plan.
- Is always tough to get up and running, but usually accelerates once you find value.

Example: MOb

1. Introduction

1.1 Purpose of This Document

This document describes the Mass Observation (MOb) acceptance test plan, which works as a guidance for MOb acceptance test activities. The acceptance test ensures that MOb product meets the customer requirements at an accredited level.

Too board

1.2 Intended Audience

This document is intended for:

- MOb testers
- MOb developers
- MOb customer
- MOb project supervisor
- All other stakeholders who are interested in this project

1.3 Scope

This document includes the plan, items, scope, approach, environment and procedure of MOb acceptance test. Then the pass/fail criteria of the test items are defined. After that, the responsibilities of developers and user representatives are identified. At last, risks and contingencies are specified to ensure the test reliability. Other information not related to the test activities is not included in this document.

IEEE 829-1998: Test Plan

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Features to be Tested
6. Features not to be Tested

What?

7. Approach
8. Item Pass/Fail Criteria
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Needs

How?

13. Staffing and Training Needs
14. Responsibilities

Who?

15. Schedule

When?

16. Planning Risks and Contingencies
17. Approvals
18. Glossary

Decisions in Test Planning

- They highlights several important decisions that shall be made during test planning.
  1. What will be tested and what will not be tested, and why?
  2. What process and method will be used?
  3. What are the deliverables generated in this project?
  4. What resources are needed?
  5. What are the responsibilities of each participants?
  6. What is the schedule?
  7. What are the risks and contingencies?

Decision #1: What Will be Tested

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Features to be Tested
6. Features not to be Tested
7. Approach
8. Item Pass/Fail Criteria
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Needs
13. Staffing and Training Needs
14. Responsibilities
15. Schedule
16. Planning Risks and Contingencies
17. Approvals
18. Glossary

What Will be Tested

- Test Items:
  - Executable code
  - Scripts
  - Library
  - User manual
  - Initialization data

- Features to be tested:
  - Logical application functions and user interfaces
    - Usually from requirement specification.
    - The users perspective.
  - Modules
    - From design spec. or build list.
    - The development perspective.

What Will Not be Tested

- List of features that will not be tested.

- Possible Reasons of NOT testing the code:
  - Low risk features ---- how determined?
  - Future release features that has finished coding phase (and unfortunately “may be” a problem if included in the code)
  - Not a high priority feature (from requirement)

Case Study: MOb

3. Test items

Based on the MOb requirements and design description, application modules of web application, mobile Android application, mobile web application, end-to-end scenario and non-functional scenario will be tested.

Missing configuration management information, such as revision number.

3.1 Web application

In web application, OE can be created, started, terminated, edited, deleted and viewed by the Initiator. Initiator will be able to manage the group of people who make observations and select people who consume the observation results. Initiator will also be able to select OE interface such as check-boxes, image capture, voice recording and written notes.

In this application, consumers who are authorized will be able to view OE results, filter by fields and optionally explore/analyze the result. Administrator is a special actor in web application who will manage all the users as initiators, observers and consumers.

Consider using bullet list.

3.2 Mobile Android application

In mobile Android application, observer is able to make an observation as prescribed by initiator, such as answering questions (posed by the Initiator), making notes, taking pictures, providing audio commentary. Observation data with a unique user ID will be time and location stamped. And the data will be transmitted to web server by instant transfer with cellular connection, delayed transfer when Wi-Fi is available, or transfer to desktop computer with cable connection.

3.3 Mobile web application

With mobile web application, observer can make an observation without Android mobile. The application is adapted with Opera Mini. Any mobile devices connecting with network is able to make observation and send data. The functionality of mobile web application is the replacement of mobile Android application to enable end users with non-Android platform with this application.

Case Study: MOb

These are features, not test items.

3.3 End-to-end Scenario

In the end-to-end scenario, initiator enters a study question as an OE in web application and makes it available to mobile observers. Then observers make an observation on mobile application by answering questions, making notes, taking pictures or recording audio. After that, the observation data will be sent to web application. And web application will collect the result and make them available for consumers to analysis and view.

3.4 Non-functional Scenario

For privacy/security and authentication, system will generate a key for the observer selected, and the key will be used to relay back the information when OE demands a signature. Only observation data with the key matched will be valid.

For accessibility, Android platform is used for mobile application device to ensure that data can be encrypted to make the transmission secure.

For affordability, mobile application will provide means to capture observations even if it hasn't got access to the Internet.

For performance, the time of loading a web application, searching the database and printing the result should be at an acceptable level and system should recovery to normal state within a specified time.

No enough information for a test plan

Case Study: MOb

4. Features to be tested

Identity	Status	Priority	Description	Source
			System Administration	
ADM-1	I	1	Ability to create a user.	SYS
			Administrator is able to create the user and assign a role to him. The role can be of initiator or observer or consumer or a combination of the above roles.	
ADM-2	I	2	Ability to edit a user.	SYS
			Administrator is able to edit properties of a user including his role.	
ADM-3	I	2	Ability to view a user.	SYS
			Administrator is able to view the properties of a user.	
ADM-4	I	1	Ability to delete a user.	SYS
			Administrator is able to delete a user. Case 1: Initiator can be deleted only after the OE created by him is deleted. Case 2: Observer is deleted. Case 3: Consumer is deleted. Case 4: Guest User (Inactive User) is deleted.	
			Initiator	
INI-1	I	1	Ability to create an observation event.	CTM
			The initiator will be able to create observation events based on certain predetermined parameters. The observation event will be identified by its own unique id, type, etc.	
INI-2	I	1	Ability to start an observation event.	CTM
			The initiator will be able to start a created OE by him. To start an OE, it should have at least one observer tagged to it and medium of observation identified	

Case Study: MOb

5. Features not to be tested

The following features shall not be tested because they are either inexecutable or uncontrollable.

- Database. Web application will perform the integrity check and all data retrieved will not be altered so database will not be tested as a standalone feature.
- Large number of users as high-load stress. High-load is only possible when the product is released to public because the emulated test result in lab is not convincible.
- Network delays. It's not possible to control the network delay as in realistic environment so it will not be covered in this acceptance test.

Case Study: Transport4You

3. Test items

The system is divided into three different components namely, Web Component, Transport Unit Component and Main Application Component.

- Web Application
- Transport Unit Application
- Transport Main Application

Better documentation with bullet list

Brief functionalities of these components are:-

Transport Web Application:

- User registration, ticket purchase, and route management is persisted to the database.

Transport Unit Application:

- Uses the WIFI or/and Bluetooth device to smartly detect users inside the TU.
- Uses the GPS device to find out TU's GPS coordinates.
- Uses the TU event system to find out when the TU arrives/leaves a station.
- Uses the GPRS device to send user information to the TMA after leaving each station.

Transport Main Application:

- Uses the GPRS device to receive user information from multiple TUA (each TU leaving a station in the city).
- Uses the database to identify the users by their WIFI/Bluetooth MAC address.
- Uses the credit card system to perform ticket purchase (if necessary).
- Uses the database to identify users habitual routes.
- Uses SMS gateway to send notifications (if necessary).

Case Study: Transport4You

4. Features to be tested

Features that will be tested are as following:

Identity	Status	Priority	Description	Test Case ID
			Transport Main Application	
F-TMA1	I	1	Application enforces billing.	TMA-1
F-TMA2	I	1	Application notifies users about payment: • On payment success. • Reminder to pay ticket. • On payment failure. • On ticket expiration.	TMA-2
F-TMA3	I	1	Application checks when ticket is expired and if user is still in vehicle application buys new ticket.	TMA-3
F-TMA4	I	1	Application records routes of every user.	TMA-4
F-TMA5	I	1	Application performs scheduled standard route identification. (Finding standard users routes based on existing data about users)	TMA-5
F-TMA6	I	1	Application notifies users about routes and suggests alternative: • Interruption in standard route. • Modification to standard route. • Optimization to standard route.	TMA-6
F-TMA7	I	2	If system crashes tickets are archived. (User will not lose its active tickets because of system crash)	TMA-7
F-TMA10	A	1	System provides transport line data pushing when changes occur to transport unit application.	TMA-8
F-TMA11	A	2	Application performs scheduled route optimization.	TMA-9
F-TMA12	A	2	Application performs scheduled route optimization notification.	TMA-10
F-TMA13	A	2	Application has structured schema for network input.	TMA-11

Case Study: Transport4You

6. Features not to be tested

SMS Gateway's and Google API's functionality will not be tested. Also some of the exceptional cases like high workload on server, network congestion and network delays will not be tested.

Has not explain why and analyze the risk.

Decision #2: Process and Approach

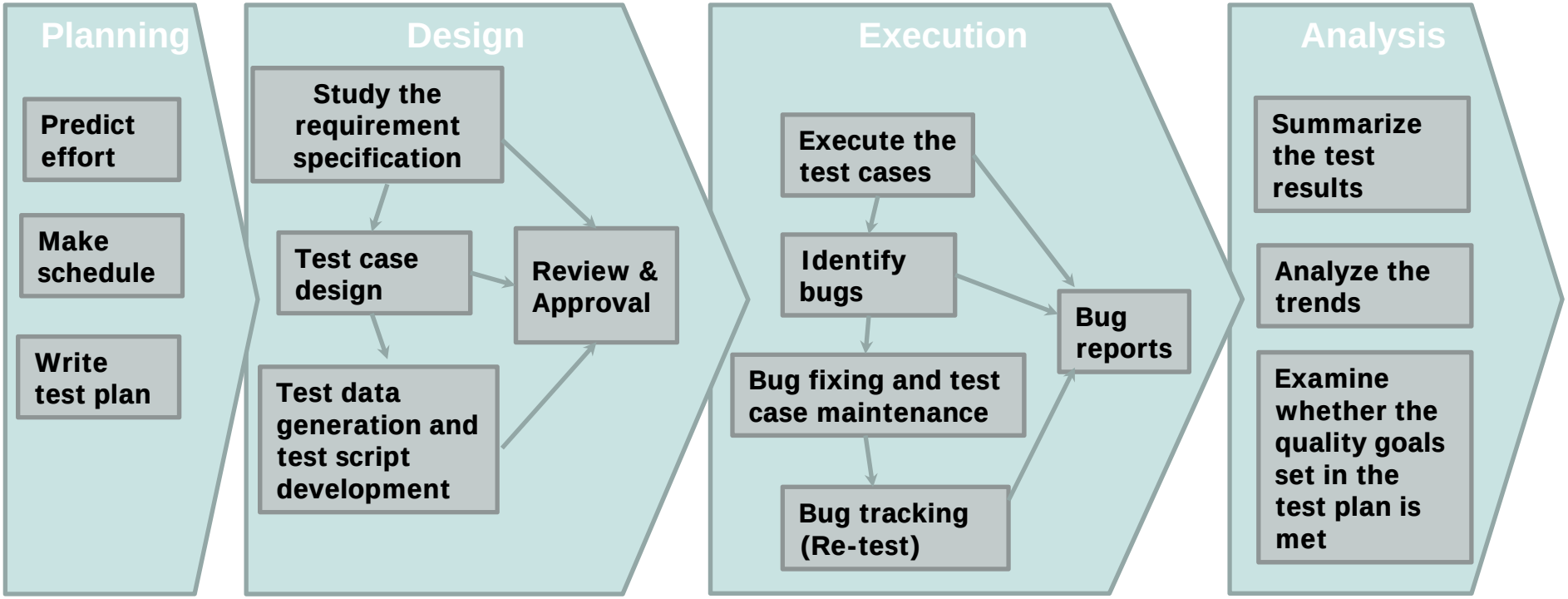
1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Features to be Tested
6. Features not to be Tested
7. Item Pass/Fail Criteria
8. Approach
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Needs
13. Staffing and Training Needs
14. Responsibilities
15. Schedule
16. Planning Risks and Contingencies
17. Approvals
18. Glossary

Process and Approach

- **Approach**: Which test adequacy criteria will be applied? Which techniques to use?
  - Blackbox testing
    - Level-1, level-2, or level-3? Mixed?
    - Decision table or category-partition testing?
  - Whitebox testing
    - Which code coverage? statement or branch? Mixed?
    - How to do the instrumentation?
    - How to generate test data?
- **Item Pass/Fail Criteria**: What data will be gathered and what metrics will be used to measure the progress of testing?
  - # of test cases and the amount of coverage (paths, code, etc.)
  - # of problems found by severity, by area, by source, by time
  - # of problems fixed, returned with no resolution, in abeyance
  - # of problems resolved by severity, turn around time, by area, etc.

Process and Approach

- **Test Tasks**: the list of tasks that are required to do.



Process and Approach

- **Suspension Criteria and Resumption Requirements**:
  - How do test engineers, whose job is to expose bug, and developer, whose job is to fix bug, synchronize their work?
    - Batch mode: test engineers stop when they find a bug, resume when the developers fix it.
    - Pipeline mode: test engineers continue to find new bugs when developers are fixing the currently reported bugs.
  - Need to define a protocol in the test plan.
- **Test deliverables**: the products of the test project, such as test plan document, test design documents, test case, test scripts, etc.
- **Test tasks:** the tasks to be done in this test project.

Case Study: MOb

6. Approach

Poorly written!

Acceptance test will be executed based on this acceptance test plan. And after all test cases are executed, a test report will be summarized to show the quality of our MOb product. Following test approaches will be used in test execution:

- **Unit test**. Developers are responsible for unit test as white-box testing. The implementation of each module and individual component will be verified separately.
- **Integration test**. After the unit test is passed above the defined quality threshold, testers will execute the integration test cases. After all the modules are integrated, it's crucial to test the product as a black-box. End-to-end scenarios will be tested to ensure the communication functionality.
- **Regression test**. After developers fix the bug in one feature, regression test will be executed by testers to ensure that the other functions are not affected.
- **Field test**. Firstly, untrained end users recreate one or more existing (but narrow) mass observation events in the MOb system. A number of observers will be invited to help with evaluation. After that, post event questionnaires will be used to collect quantitative usage data as well as qualitative data and further improvement will be taken into consideration.
- **Positive and negative testing design technique**. This approach will be combined with unit test and integration test. Test cases are designed in sunny day scenarios, which ensure that all functional requirements are satisfied. What's more, rainy day test cases will also be covered to show how the system reacts with invalid operations.

Case Study: MOb

Poorly written!

7. Item pass/fail criteria

Details of the test cases are specified in section 10. Following the principles outlined below, a test item would be judged as pass or fail.

- Preconditions are met
- Inputs are carried out as specified
- The result works as what specified in output => Pass
- The system doesn't work or not the same as output specification => Fail

7.1 Installation and Configuration

Installation and configuration of web application does not affect the testing as long as the test environment is as required in section 9. Nevertheless, the mobile Android application should be installed successfully on the mobile with Android platform and cellular connection or internet is required, or the mobile without Android platform should have cellular or internet connection, otherwise, the integration test cannot be executed.

7.2 Documentation problems

Requirement definition document should be kept up-to-date, or there will be a waste of time for developers and testers with extra efforts on fake errors.

Test plan document should describe test cases clearly without ambiguity, which helps tester to execute the test cases and validate the requirements.

User manual document should be finished completely before filed test, or untrained end user will have trouble in using the MOb product correctly.

Case Study: MOb

8. Suspension criteria and resumption requirements

For website interface, the response time is short. Any bugs found can be fixed by developers quickly and no need to start the testing process from the beginning. However, when major bugs such as the communication between mobile and web application occur, it will block the following test cases and the testing has to be paused. The test will restart from the very beginning until the major error is solved.

A simple but sufficient protocol

Case Study: Transport4You

It seems that the author has nothing to say.

6. Approach

When a module is developed, unit tests are performed on the module by the developer. Integration testing is performed before integrating any new module with the existing modules. Finally, with the end of the project development, system testing is done to verify that all the functionalities that are integrated, works correctly.

6.1 Approach to configuration and installation

Transport4You1 testing team is responsible for setting up environment needed to run the application. Installation manual is provided by the developers and is used to install and configure the application.

7. Item pass/fail criteria

7.1 Installation and Configuration

If application is installed and configured as per the manual given by developers, application will be working fine.

7.2 Documentation problems

NA

Decision #3: Resource

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Features to be Tested
6. Features not to be Tested
7. Item Pass/Fail Criteria
8. Approach
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Needs
13. Staffing and Training Needs
14. Responsibilities
15. Schedule
16. Planning Risks and Contingencies
17. Approvals
18. Glossary

Test Resources

- Based on the amount of testing items and testing process defined so far, estimate the people resources needed
  - The skills of the people
  - Additional training needs
  - # of people
- The non-people resources needed
  - Tools
    - Configuration management
    - Automated test tool
    - Test script writing tool
  - Hardware, network, database
  - Environment (access to: developers, support personnel, management personnel, etc.)

Case Study: MOb

9. Environmental needs

9.1 Hardware

- Laptop or Desktop PC
- Mobile with Android platform

9.2 Software

- PC Operating System:
  - Windows XP (32-bit)
  - Windows Vista (32- or 64-bit)
  - Windows 7 (32- or 64-bit)
- Browsers: (screen resolutions of 1024x768px or more)
  - Opera 10 or later
  - Microsoft Internet Explorer 7 or later
  - Mozilla Firefox 3 or later
  - Apple Safari 4 or later
  - Google Chrome 4.0 or later
  - Opera Mini 10+ as mobile web browser
- Mobile Platform: Android 2.0/2.1 (Eclair) Based on Linux Kernel 2.6.29 or later
- Eclipse 3.4 (Ganymede) or 3.5 (Galileo) with ADT Plugin

9.3 Other

- Internet connection for PC
- WiFi/3G/GPRS connection for mobile

Decision #4: Tasks Assignment

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Features to be Tested
6. Features not to be Tested
7. Item Pass/Fail Criteria
8. Approach
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Needs
13. Staffing and Training Needs
14. Responsibilities
15. Schedule
16. Planning Risks and Contingencies
17. Approvals
18. Glossary

Task Assignment

- Assign personal for each tasks.

Table 17.1. High-Level Tester Assignments for WordPad

Tester	Test Assignments
Al	Character formatting: fonts, size, color, style
Sarah	Layout: bullets, paragraphs, tabs, wrapping
Luis	Configuration and compatibility
Jolie	UI: usability, appearance, accessibility
Valerie	Documentation: online help, rollover help
Ron	Stress and load

Another Example

任务(–:责任人，*:参与人)	程序管理员	程序员	测试员	技术文档作者	营销人员	产品支持人员的任务
1）撰写产品版本声明	*				–	
2）创建产品组成部分清单	_					
3）创建合同	–				*	
4）产品设计/功能划分	_			*	*	
5）项目总体进度	–	*		*	*	
6）制作和维护产品说明书	_					
7）审查产品说明书—6	*	*	*	*	*	*
8）内部产品的体系结构	*	–				
9）设计和编写代码		–				
10）测试计划			–			
11）审查测试计划		*	–	*	*	*
12）单元测试		–	*			
13）总体测试			–			
14）创建配置清单		*	–		*	*
15）配置测试			–			
16）定义性能基准	–		*			
17）运行基准测试			–			
18）内容测试			*	–		
18）来自其它团队的测试代码			*			
19）自动化/维护构建过程		–				
20）磁盘构建/复制		–				
21）磁盘质量保证			–			
22）创建beta测试清单					–	*
23）管理beta程序	*		*		–	*
24）审查印刷的资料	*	*	*	–	*	*
25）定义演示版本	*				–	
26）生成演示版本	*				–	
27）测试演示版本			–			
28）缺陷会议	–	*	*		*	*

Case Study: MOb

Too board

11. Responsibilities

11.1 Developers

- Unit test when implement their owned models
- Fix the bug and make sure the code change does not affect other functions
- Deliver a new testable version to tester

11.2 Testers

- Write acceptance test plan
- Execute test cases and record test results
- Inform developers with the bugs as early as possible
- Summarize the results as Test Report

11.3 User representative

- Clarify the suspicious requirements
- Comment on the test plan and test cases

Decision #5: Schedule

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Features to be Tested
6. Features not to be Tested
7. Item Pass/Fail Criteria
8. Approach
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Needs
13. Staffing and Training Needs
14. Responsibilities
15. Schedule
16. Planning Risks and Contingencies
17. Approvals
18. Glossary

Test Schedule

- Based on:
  - Amount of test items
  - Test process and methodology utilized
  - Number of estimated test cases
  - Estimated test resources

- A schedule containing the following information should be stated:
  - Tasks and Sequences of Tasks
  - Assigned persons
  - Time duration

Examples

Table 17.2. A Test Schedule Based on Fixed Dates

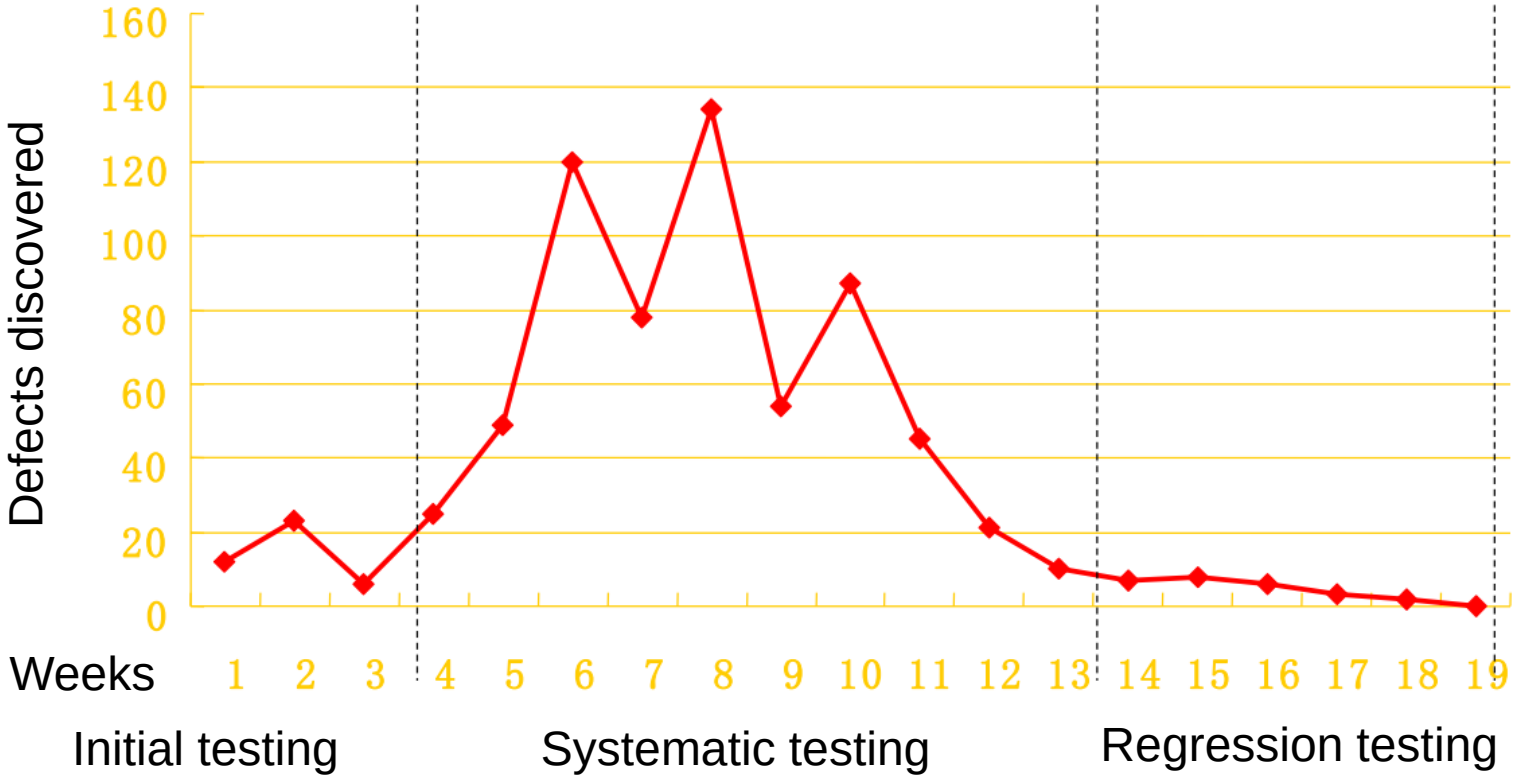
Testing Task	Date
Test Plan Complete	3/5/2001
Test Cases Complete	6/1/2001
Test Pass #1	6/15/20018/1/2001
Test Pass #2	8/15/200110/1/2001
Test Pass #3	10/15/200111/15/2001

Table 17.3. A Test Schedule Based on Relative Dates

Testing Task	Start Date	Duration
Test Plan Complete	7 days after spec complete	4 weeks
Test Cases Complete	Test plan complete	12 weeks
Test Pass #1	Code complete build	6 weeks
Test Pass #2	Beta build	6 weeks
Test Pass #3	Release build	4 weeks

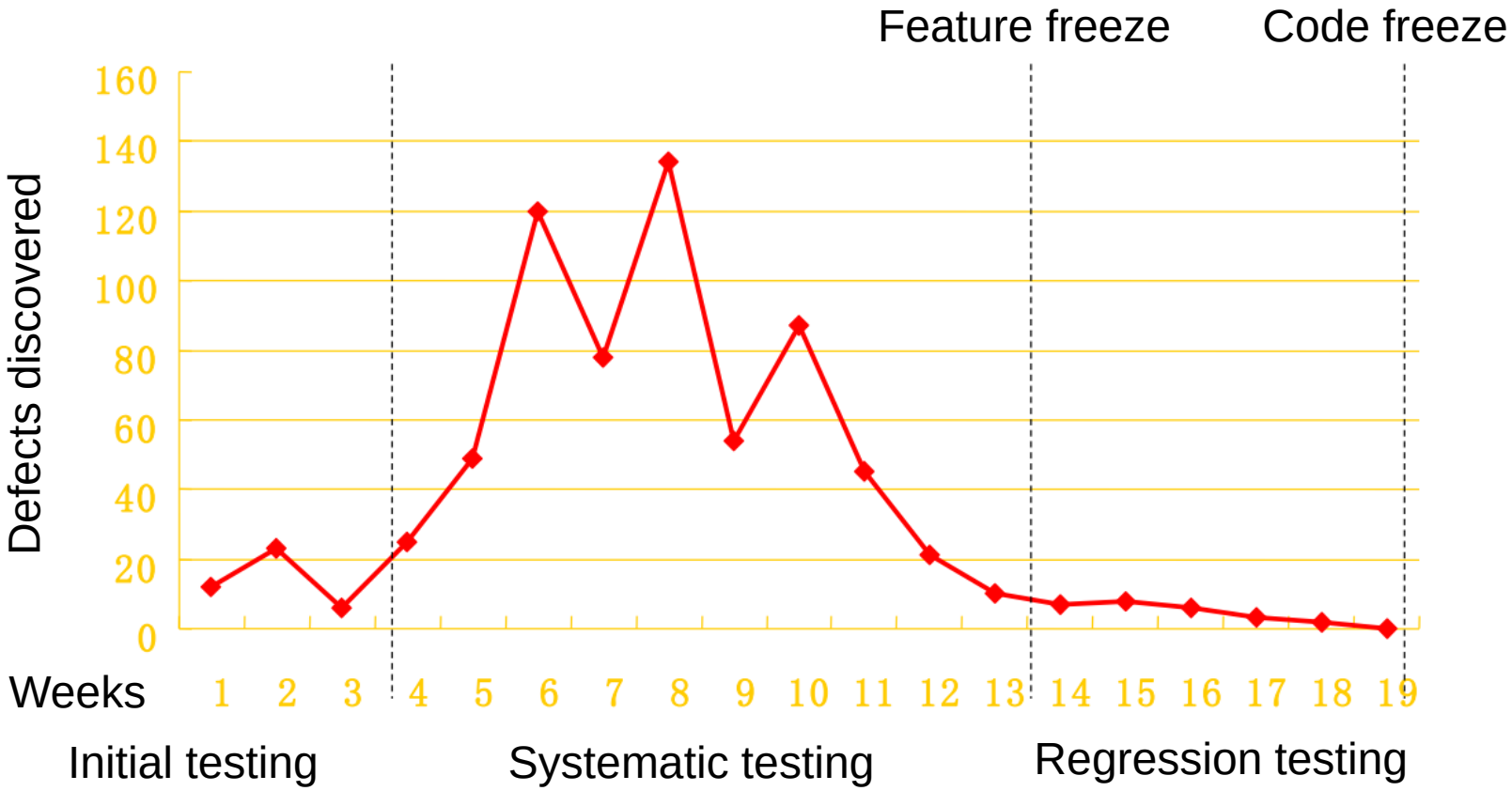
Phases of Testing

- Initial testing: cover key features and execution paths
- Systematic testing: go for completeness
- Regression testing: re-execute existing test cases



Feature Freeze and Code Freeze

- Two major milestones in the test schedule
  - No more features shall be added.
  - No more modification to the code shall be done.



Decision #6: Risks and Contingencies

1. Test Plan Identifier
2. References
3. Introduction
4. Test Items
5. Features to be Tested
6. Features not to be Tested
7. Item Pass/Fail Criteria
8. Approach
9. Suspension Criteria and Resumption Requirements
10. Test Deliverables
11. Test Tasks
12. Environmental Needs
13. Staffing and Training Needs
14. Responsibilities
15. Schedule
16. Planning Risks and Contingencies
17. Approvals
18. Glossary

Risks and Contingencies

- Examples of Risks:
  - Changes to original requirements or design
  - Lack of available and qualified personnel
  - Lack of (or late) delivery from the development organization:
    - Requirements
    - Design
    - Code
  - Late delivery of tools and other resources
- State the contingencies if the Risk item occurs
  - Move schedule
  - Do less testing
  - Change the goals and exit criteria

Case Study

MOb

12. Risks and contingencies

We have tried to test on various browsers as much as possible, but it's impossible to test on all the browsers. What's more, mobile Android application is tested on Android devices, and mobile web application is tested on limited mobiles, thus we cannot predict the system behavior on the other mobile platforms (eg. iPhone, Blackberry, Symbian platform etc.). Further investigation is required to verify and improve this MOb product.

Transport4U

12. Risks and contingencies

N/A

Final Remark

- *A test plan is a valuable tool to the extent that it helps you manage your testing project and find bugs. Beyond that, it is a diversion of resources.* Kaner et al.

- In fact, this wisdom applies to any documentation activities.

IEEE 829-2008: Master Test Plan

1. Introduction
1.1. Document identifier
1.2. Scope
1.3. References
1.4. System overview and key features
1.5. Test overview
- 1.5.1 Organization
- 1.5.2 Master test schedule
- 1.5.3 Integrity level schema
- 1.5.4 Resources summary
- 1.5.5 Responsibilities
- 1.5.6 Tools, techniques, methods, and metrics

2. Details of the Master Test Plan
2.1. Test processes including definition of test levels
- 2.1.1 Process: Management
  - 2.1.1.1 Activity: Management of test effort
- 2.1.2 Process: Acquisition
  - 2.1.2.1: Activity: Acquisition support test
- 2.1.3 Process: Supply
  - 2.1.3.1 Activity: Planning test
- 2.1.4 Process: Development
  - 2.1.4.1 Activity: Concept
  - 2.1.4.2 Activity: Requirements
  - 2.1.4.3 Activity: Design
  - 2.1.4.4 Activity: Implementation
  - 2.1.4.5 Activity: Test
  - 2.1.4.6 Activity: Installation/checkout
- 2.1.5 Process: Operation
  - 2.1.5.1 Activity: Operational test
- 2.1.6 Process: Maintenance
  - 2.1.6.1 Activity: Maintenance test

2.2. Test documentation requirements
2.3. Test administration requirements
2.4. Test reporting requirements

IEEE 829-2008: Level Test Plan

1. Introduction
1.1. Document identifier
1.2. Scope
1.3. References
1.4. Level in the overall sequence
1.5. Test classes and overall test conditions

2. Details for this level of test plan
2.1 Test items and their identifiers
2.2 Test Traceability Matrix
2.3 Features to be tested
2.4 Features not to be tested
2.5 Approach
2.6 Item pass/fail criteria
2.7 Suspension criteria and resumption requirements
2.8 Test deliverables

3. Test management
3.1 Planned activities and tasks; test progression
3.2 Environment/infrastructure
3.3 Responsibilities and authority
3.4 Interfaces among the parties involved
3.5 Resources and their allocation
3.6 Training
3.7 Schedules, estimates, and costs
3.8 Risk(s) and contingency(s)

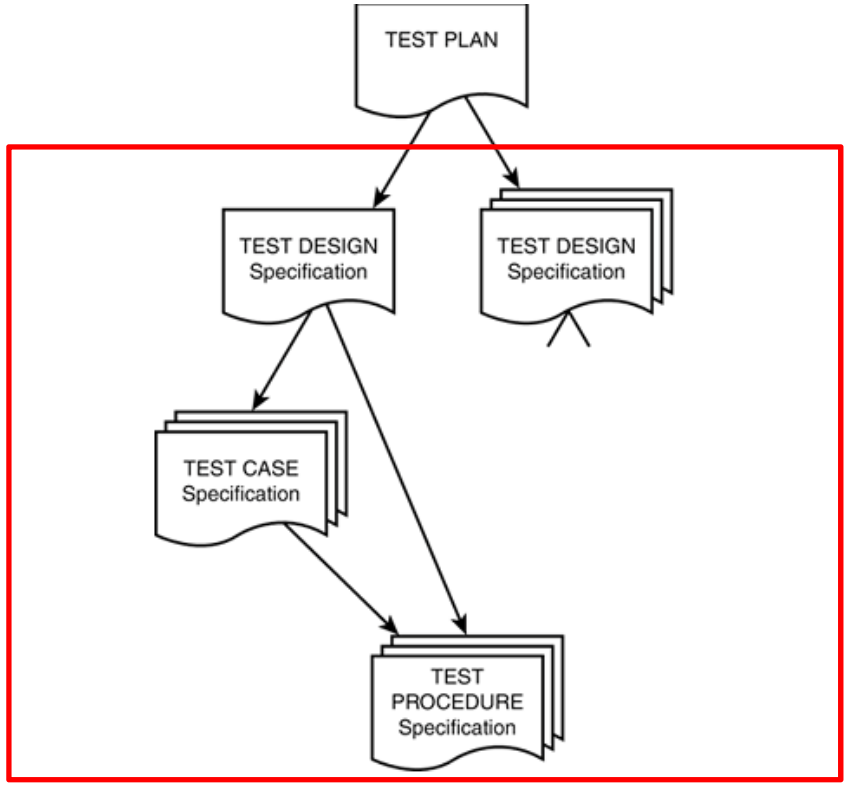
4. General
4.1 Quality assurance procedures
4.2 Metrics
4.3 Test coverage
4.4 Glossary
4.5 Document change procedures and history

Understanding IEEE 829

- Test Planning（如何做测试计划）
  - IEEE 829 Document: Test Plan
- Test Design （如何做测试设计）
  - IEEE 829 Documents: Test Design Specification, Test Case Specification, Test Procedure Specification
  - Tool: TestLink
- Test Report（如何编写测试报告）
  - IEEE 829 Documents: Test Log, Test Incident Report, Test Summary Report

Test Design Overview

- The overall project test plan is written at a very high level. It breaks out the software into specific features and testable items and assigns them to individual testers, but it doesn't specify exactly how those features will be tested.



Test Design

1. Test Design Specification
2. Test Case Specification
3. Test Procedure Specification

1. Test Design Specification

- Refine the test approach and generate test conditions
  - e.g. test frames in TSL
- Determine Pass \ Fail Criteria.
  - Test oracle

- IEEE 829-1998 template:
  1. Identifiers.
  2. Features to be tested.
  3. Approach.
  4. Test case identification.
  5. Pass/fail criteria.

IEEE 829-2008: Almost the Same

Level Test Design Outline (full example)

1. Introduction

1.1. Document identifier

1.2. Scope

1.3. References

2. Details of the Level Test Design

2.1. Features to be tested

2.2. Approach refinements

2.3. Test identification

2.4. Feature pass/fail criteria

2.5 Test deliverables

3. General

3.1. Glossary

3.2. Document change procedures and history

Test Design Specification Structure

- **Features to be Tested** identifies test items and describes features and combinations of features that are the object of this design specification. Reference on functional specification for each feature or combination of features should be included.

- **Approach Refinements** section describes the following:
  - Specific test techniques to be used for testing features or combinations of features
  - Whether automation of test cases will be provided or not
  - Any other information which describes approach to testing

- **Feature Pass/Fail Criteria** specifies the criteria to be used to determine whether the feature or feature combination has passed or failed

Test Design Specification Structure

- **Test Identification** section describe what the test cases do.

Example:

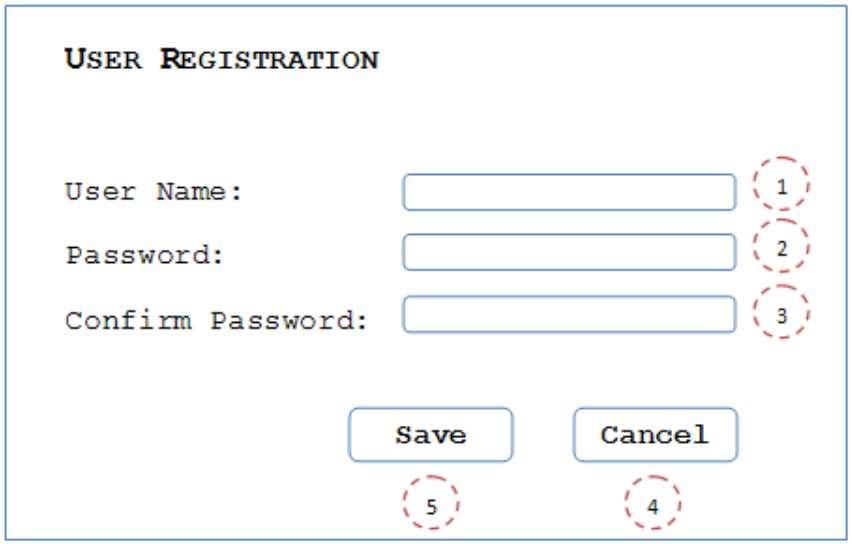
Test Identification

Configuration

Functional Requirements	#	Test Case	Description
Server profile is used for organization of correct connection to application server	1.	Verify that server profile can be configured	This test case verifies that user can configure server profile, for appropriate version of WebLogic.
Verify connection to the Application Server	2.	Verify connection in case AS is located on remote machine	This test case verifies that we can simply connect to appropriate application server which located on remote machine.
	3.	Verify connection in case application and AS is located on local machine	This test case verifies that we can simply connect to appropriate application server which located on our local machine.
	4.	Verify correct work of each functionality when server isn't picked up	This test case verifies that there is no possibility to use each functionality in case server isn't started and appropriate error message is appeared

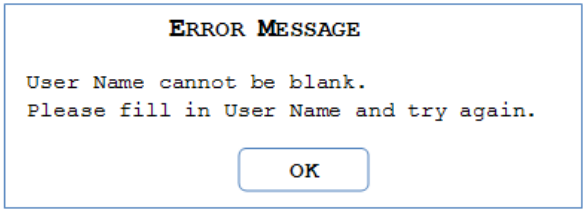
Case Study: User Registration

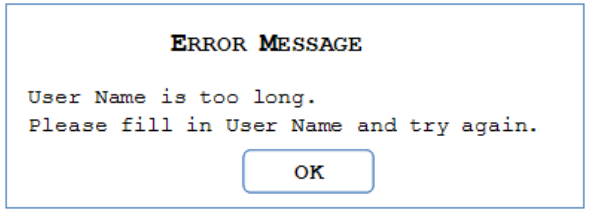
- **Feature:** Administrator user should be able to create a simple user account to log in application.
- **Functional Requirements:** 'User Registration' page should contain three fields 'User Name', 'Password', 'Confirm Password' and two buttons – 'Save' and 'Cancel'.
- **GUI:**

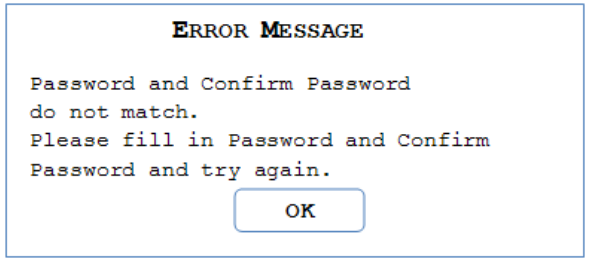


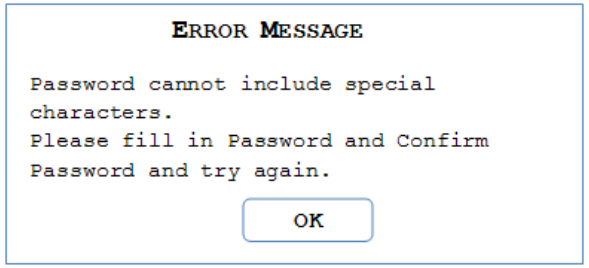
1. 'User Name' field is limited by 10 symbols and should contain letters of Latin alphabet only. 'User Name' field is empty by default. User Name should be unique in the system.
2. 'Password' field should be no less than 4 symbols long and should include only numbers and letters of Latin alphabet only. 'Password' field is empty by default.
3. 'Confirm Password' field should be equal to 'Password'. 'Confirm Password' field is empty by default.
4. 'Cancel' button cancels account creation and closes 'User Registration' page.
5. 'Save' button validates data entered into fields on 'User Registration' page and creates user account if entered data are correct; or shows error dialogs if validation fails. Validation should be provided in following order: User Name, Password, and Confirm Password.

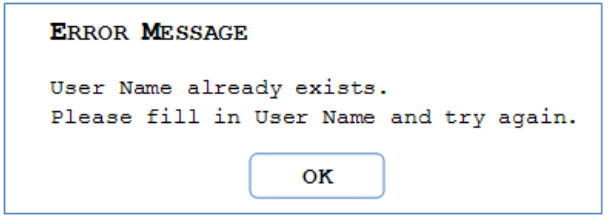
Case Study: User Registration











Test Design Specification

Features	Test Case Identification	Description
'Save' button functionality	Creating new user account and save	This test verifies that user account could be created if all fields on 'User Registration' page are filled with correct data; and 'User Registration' page is closed on save action
'Cancel' button functionality	Creating new user account and cancel	This test verifies that user account is not created after filling in fields on 'User Registration' page and canceling; and 'User Registration' page is closed on cancel action
Default values	Default values on the 'User Registration' page	This test verifies that all fields on 'User Registration' page are blank by default
'User Name' field validation	Error dialog on saving user account with too long user name	This test verifies that error dialog appears while save action if user name length is too long: 1)boundary length – 11 characters 2)restricted length – more than 11 characters
	Error dialog on saving user account with blank 'User Name' field	This test verifies that error dialog appears while save action if 'User Name' field is blank
	Verify boundary length for user name	This test verifies that user account having user name with boundary length 1 or 10 could be created
	Error dialog on saving user account with wrong user name	This test verifies that error dialog appears while save action if 'User Name' field include: 1)special symbols; 2)numbers; 3)both
	Error dialog on saving already existing user account	This test verifies that error dialog appears while save action if user already exists in the system
'Password' field validation	Error dialog on saving user account with too short password	This test verifies that error dialog appears while save action if password length is too short: 1)boundary length – 3 characters 2)restricted length – less than 3 characters
	Error dialog on saving user account with blank 'Password' field	This test verifies that error dialog appears while save action if password is blank
	Verify boundary length for password	This test verifies that user account having password with boundary length 4 could be created
	Error dialog on saving user account with incorrect password	This test verifies that error dialog appears while save action if 'Password' field includes special symbols
'Confirm Password' field validation	Error dialog on saving user account with unequal password and confirm password	This test verifies that error dialog appears while save action if: 1)'Confirm Password' field is blank 2)password and confirm password do not match

Can use TSL or cause-effect graph as a more formal procedure

2. Test Case Specification

- IEEE 829 states that the test case specification "documents the actual values used for input along with the anticipated outputs. A test case also identifies any constraints on the test procedure resulting from use of that specific test case.“

- IEEE 829-1998 template:
  1. Identifiers.
  2. Test item.
  3. Input specification.
  4. Output specification.
  5. Environmental needs.
  6. Special procedural requirements.
  7. Intercase dependencies.

Example

10.1.2.2 SM_IMPFILE_TC_002

Description:
To verify the functionality of saving a project with the routes, stations and background image

Test type:
Positive

Preconditions:
The user should have provided any input to the application or no input at all.

Input definition:

1. Click on File and select the "Save project" option
2. Navigate to the desired location where the file should be saved
3. Enter the name of a file and click on the "Save project" button

Output definition:

1. File dialog must appear with files and directories from local file system shown
2. File dialog must show default naming for a file "SMProject1.mdm"
3. The file must be saved in the specified location. This can be verified by browsing manually to the location where the file was supposed to save. If there was a file with the same name as specified, then File dialog must prompt a user whether an existing file should be overwritten or not.

Remarks:
The validity of the saved file can be checked by referring to the test case SM_IMPFILE_TC_001 which tests the opening of an existing file.

3. Test Procedure Specification

- IEEE 829 states that the test procedure specification "Identifies all the steps required to operate the system and exercise the specified test cases in order to implement the associated test design."
  - Also known as *test script specification* if test automation is used.

- The test procedure or test script specification defines the step-by-step details of exactly how to perform the test cases.

Test Case Planning Overview

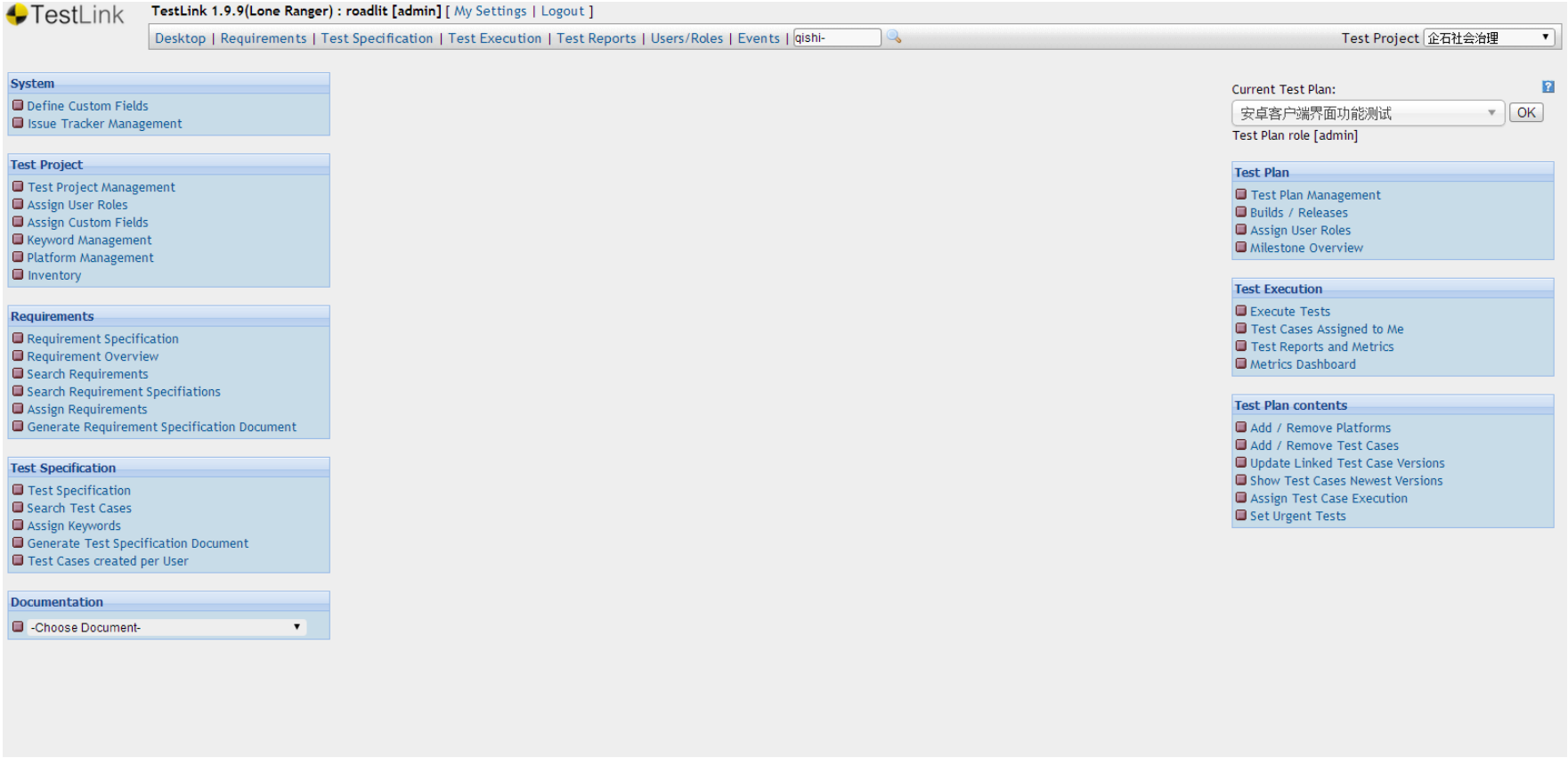
- IEEE 829-1998 template:
  - Identifier.
  - Purpose.
  - Special requirements.
  - Procedure steps.
    - Log.
    - Setup.
    - Start.
    - Procedure.
    - Measure.
    - Shut down.
    - Restart.
    - Stop.
    - Wrap up.
    - Contingencies.

Final Remark

- Criteria of a good test design
  - Test cases are well-organized.
  - Test execution is repeatable.
  - Test cases are traceable.
  - Test results are verifiable.
- Check whether the following questions can be answered:
  - Which test cases do you plan to run?
  - How many test cases do you plan to run? How long will it take to run them?
  - Can you pick and choose test suites (groups of related test cases) to run on particular features or areas of the software?
  - When you run the cases, will you be able to record which ones pass and which ones fail?
  - Of the ones that failed, which ones also failed the last time you ran them?
  - What percentage of the cases passed the last time you ran them?

Tool: TestLink

- At http://eden.sysu.edu.cn/testlink



TestLink:

- Test project management
  - 人员管理
- Test requirement management
  - 测试需求管理
- Test specification management
  - 测试规范管理
- Test plan management
  - 测试（执行）计划管理
- Test execution management
  - 测试执行管理

Using TestLink

- Step 1: Create a project
- Step 2: Manage the requirement
- Step 3: Create a test specification
- Step 4: Design test suites and test cases
- Step 5: Create a test plan
- Step 6: Assign test cases
- Step 7: Execute test cases
- Step 8: Generate test report

About Roles

TestLink系统提供了六种角色，相对应的功能权限如下：

- Guest:可以浏览测试规范、关键词、测试结果以及编辑个人信息；
- Tester:可以浏览测试规范、关键词、测试结果以及编辑测试执行结果；
- Test Designer:编辑测试规范、关键词和需求规约；
- Senior Tester:允许编辑测试规范、关键词、需求以及测试执行和创建发布；
- Leader:允许编辑测试规范、关键词、需求、测试执行、测试计划（包括优先级、里程碑和分配计划）以及发布；
- Admin:一切权力，包括用户管理；

Understanding IEEE 829

- Test Planning（如何做测试计划）
  - IEEE 829 Document: Test Plan
- Test Design （如何做测试设计）
  - IEEE 829 Documents: Test Design Specification, Test Case Specification, Test Procedure Specification
  - Tool: TestLink
- Test Report（如何编写测试报告）
  - IEEE 829 Documents: Test Incident Report and Test Summary Report
  - Tool: Bugzilla

Test Incidence Report

- Fundamental principles for reporting a bug:
  - Report bugs as soon as possible.
  - Effectively describe the bugs.
    - Minimal.
    - Singular.
    - Obvious and general.
    - Reproducible.
  - Be non-judgmental in reporting bugs.
- IEEE 829-1998 template:
  1. Identifier.
  2. Summary.
  3. Incident Description.
  4. Impact.

3.1	输入
3.2	期望得到的结果
3.3	实际结果
3.4	异常情况
3.5	日期和时间
3.6	软件缺陷发生步骤
3.7	测试环境
3.8	再现测试
3.9	测试人员
3.10	见证人

软件缺陷报告文档

公司名称：_____ BUG 报告：_____ BUG# ：_____

软件：_____ 版本：_____

测试员：_____日期：_____

严重性：_____优先级：_____ 是否会重现：是 否

标题：_____

描述：_____

_____

_____

解决方法：_____

解决日期：_____ 解决人：_____ 版本号：_____

解决描述：_____

_____

_____

重新测试人：_____测试版本号：_____测试日期：_____

重新测试描述：_____

_____

_____

签名：

策划：_____测式：_____

编程：_____项目管理：_____

销售：_____技术支持：_____

Bug-Tracking Systems

- The bug-reporting process is complex. It requires a great deal of information, a high level of detail, and a fair amount discipline to be effective.

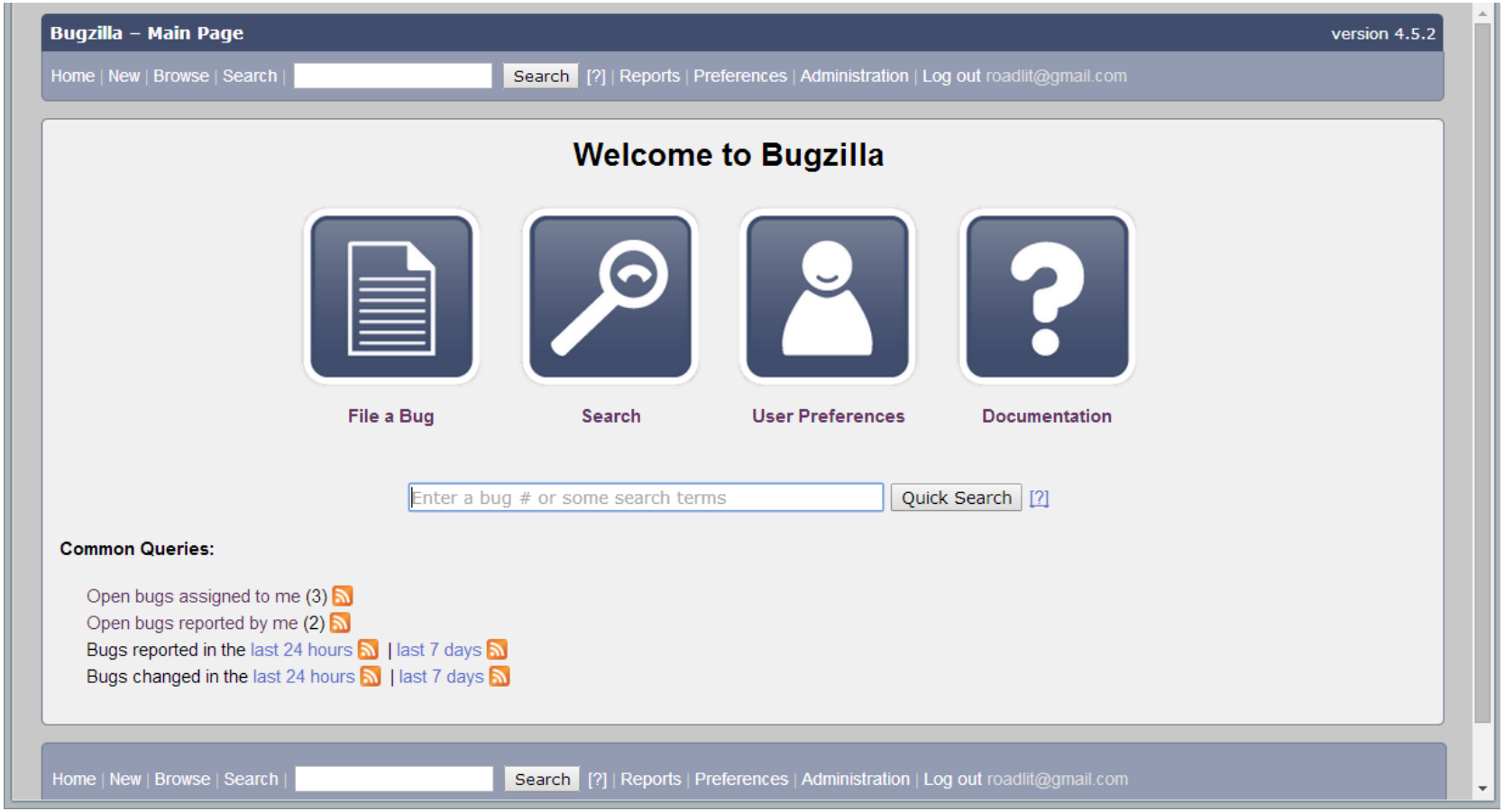
- A bug-tracking system that allows you to log the bugs you find and monitor them throughout their life cycle.
  - Can tell you what has happened in the past, what's happening now, and allows you to look at the data to make an educated guess of the future.

Bug-Tracking Systems

- A bug-tracking database can become such a fundamental means for measuring a project's status and answering following important questions.
  - What areas of the software you're testing have the most bugs? The fewest bugs?
  - How many resolved bugs are currently assigned to Martha?
  - Bob is leaving for vacation soon. Will he likely have all his bugs fixed by then?
  - Which software tester has entered the most bugs?
  - Can you please bring a list of all the open Priority 1 bugs to the project review meeting?
  - Does the software look like it's on track to meet the scheduled release date?

Tool: Bugzilla

- At http://eden.sysu.edu.cn/bugzilla



Severity/Frequency/Priority

Some definitions:

- **Severity**: the impact or consequence of the bug to the end-user, an organization, third parties, a service, etc.
- **Frequency**: the likelihood of the bug occurring or manifesting itself to the end-user, an organization, third parties, a service, etc.
- **Priority**: the relative importance of addressing and resolving the bug

- Priority would best be defined as a result of understanding the combination of both Severity and Frequency of a bug.

Defining Severity

***Severity**: the impact or consequence of the bug to the end-user, an organization, third parties, a service, etc.*

- **Blocker**
- **Critical**
- **Major**
- **Normal**
- **Minor**
- **Trivial**
- **Enhancement**

Severity - Blocker

- Blocks development and/or testing work (new build, with fix, needed immediately)

Severity - Critical

Critical:

- Application crashes
- Application or service does not work as intended; impossible to use the application or a main function of the application; critical functionality lost
- Stored data corrupted
- Conflicts with a high-priority (must-have) *accepted* requirement
- Severe memory leak
- Compromises Stanford's standards or policy (e.g. security)
- Help file is missing
- Help file is incorrect and misleads the user

Severity - Major

- Operation is significantly impacted
- User is possibly lead to creating future problems
- Conflicts with a medium-priority *accepted* requirement
- Spelling or grammatical mistakes in the User Interface

Severity - Normal

(Would like to see this changed to 'Average')

- Loss of function, but there is a clear work-around
- Unfriendly behavior that is hindering, but workable, for the user or service
- Explanation of feature missing from Help File

Severity - Minor

Cosmetic problems in the UI, other than spelling and grammatical mistakes; i.e. font sizes, placement of controls (edit boxes, list boxes, etc.) that are not hindering

Unfriendly behavior that is merely annoying to the user

A problem of small impact to the user

Spelling or grammatical mistakes in the Help File

Severity – Trivial/Enhancement

Trivial: *(Remove: Redundant to Minor. We do not need this level of granularity.)*

- (Do not use – Use Minor instead)

Enhancement:

- Nice to have feature (not in scope)
- Request for change

Severity

- Severity is a required field.
- By default, Bugzilla has ‘Normal’ selected on the new bug entry form. If this is not correct for the bug which is being entered, you should change it to the correct Severity.
- If there are better ways to describe the severity of defects for applications that are background processes (i.e. the Regis apps and Slogs), then we should be sure to add those here.

Defining Priority

***Priority**: the relative importance of addressing and resolving the defect*

- *P1: Resolve Immediately*
- *P2: Give High Attention*
- *P3: Normal Queue*
- *P4: Low Priority*
- *P5: Waiting Priority (not in scope or significant enough to be prioritized)*

Defining Priority

	Always	Likely	Unlikely
Blocker	P1	P1	P1
Critical	P1	P1	P2
Major	P1	P2	P2
Average	P2	P3	P3
Minor/Trivial	P3	P4	P4
Enhancement			

Test Summary Report: IEEE 829

1. Test identification, site, schedule and participation

1.1 The tested software identification (name, version and revision)

1.2 The documents providing the basis for the tests (name and version for each document)

1.3 Test site

1.4 Initiation and concluding times for each testing session

1.5 Test team members

1.6 Other participants

1.7 Hours invested in performing the tests

2. Test environment

2.1 Hardware and firmware configurations

2.2 Preparations and training prior to testing

Test Summary Report: IEEE 829

3. Test results

3.1 Test identification

3.2 Test case results (for each test case individually)

4. Total number of errors, their distribution and types

4.1 Summary of current tests

4.2 Comparison with previous results (for regression test summaries)

5. Special events and testers' proposals

5.1 Special events and unpredicted responses of the software during testing

5.2 Problems encountered during testing.

5.3 Proposals for changes in the test environment, including test preparations

5.4 Proposals for changes or corrections in test procedures and test case files

Metrics

- Defect Analysis
  - Defect Density: defects/lines of code
  - Defect Distribution
- Planning and Progress tracking
  - Is the software project making progress?
  - Will it be ready to release on schedule?
  - What's the risk of it hitting that date?
  - What's the overall reliability?

Thank you!