HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Software Quality Assurance
Đảm bảo chất lượng phần mềm

Lecture 7: Testing techniques and tools

Contents

- Regression Testing
- Exploratory Testing
- Agile testing
- Automation testing

7.1. Regression Testing

Regression Testing

- Regression testing is applied to code immediately after changes are made
- Goal: to assure that the changes have not had unintended consequences on the behaviour of the test object
- Apply regression testing during
  - Development phase
  - After system have been upgraded or maintainted
- Regression testing is very important to monitor the effect of changes

Why use regression testing?

- Good reasons:
    - Bug fixes often break other things the developer isn't concentrating on
    - Somtimes bug fixes don't fix the bug
    - Checking software still runs after making a change
    - Discovery faulty localization
    - Errors in build process
    - Conforming to standard or regulators
- Bad reasons:
    - Arguments in terms of replicability of results
    - Arguments in terms of quality in analogy with a production line

Risks of change

- **Bug regression testing**: checks that a bug fix has removed the symptoms of the bug that have been identified
- **Old fix regression**: checks that a new fix has not broken an old fix: refactoring should limit this as old fixes are refactored into the code.
- **Functional regression**: new code or fix has not broken previously working code.
- **Incremental Regression testing**: regression testing as we develop.
- **Localisation Testing**: tests if a product has been correctly localised for a particular market.
- **Build Testing**: has an error been introduced in the field that means the system will not build correctly.

Motivation for reusing tests

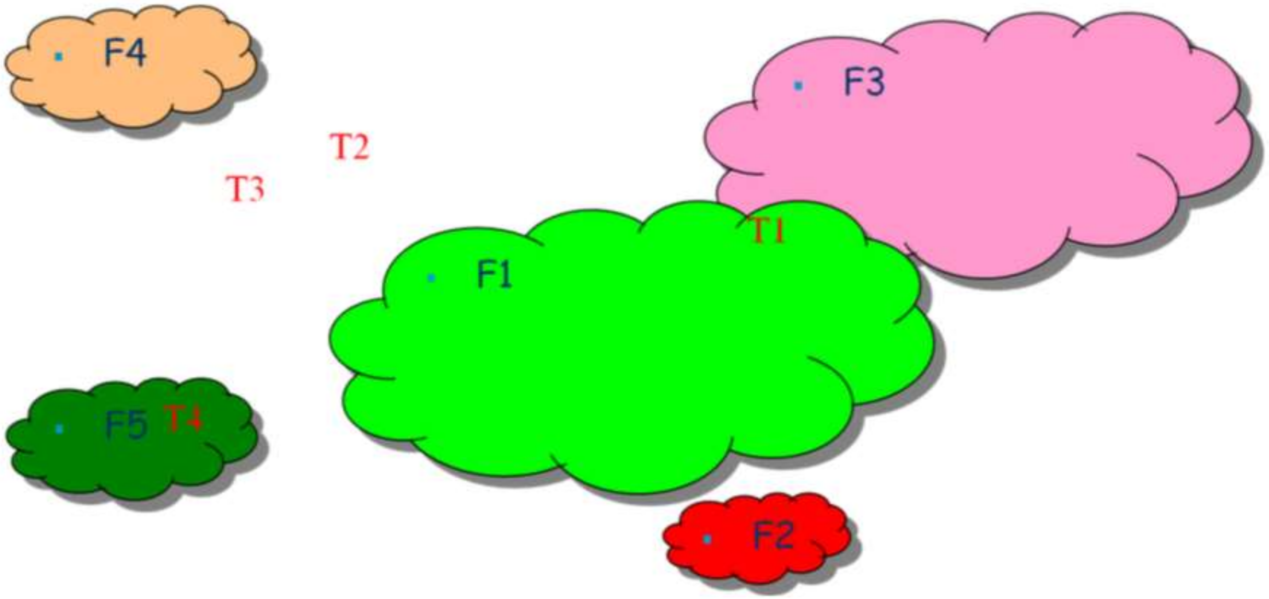
- In development (e.g. XP) tests play the role of specifications so we want to keep them fixed and reduce the cost of regression.
- In an established product:
  - Using the same tests may help us manage risk since we can focus tests on mitigating a particular risk.
  - Some tests are good at uncovering likely errors so we want to reuse.
- There may be economic motivations:
  - Automated retest (replay or oracle).
  - Replay with human inspection may reduce the need for specialist technical time (e.g. in GUI testing – this is a particularly common approach). The aim is to routinise repeat testing.

Key questions of reuse

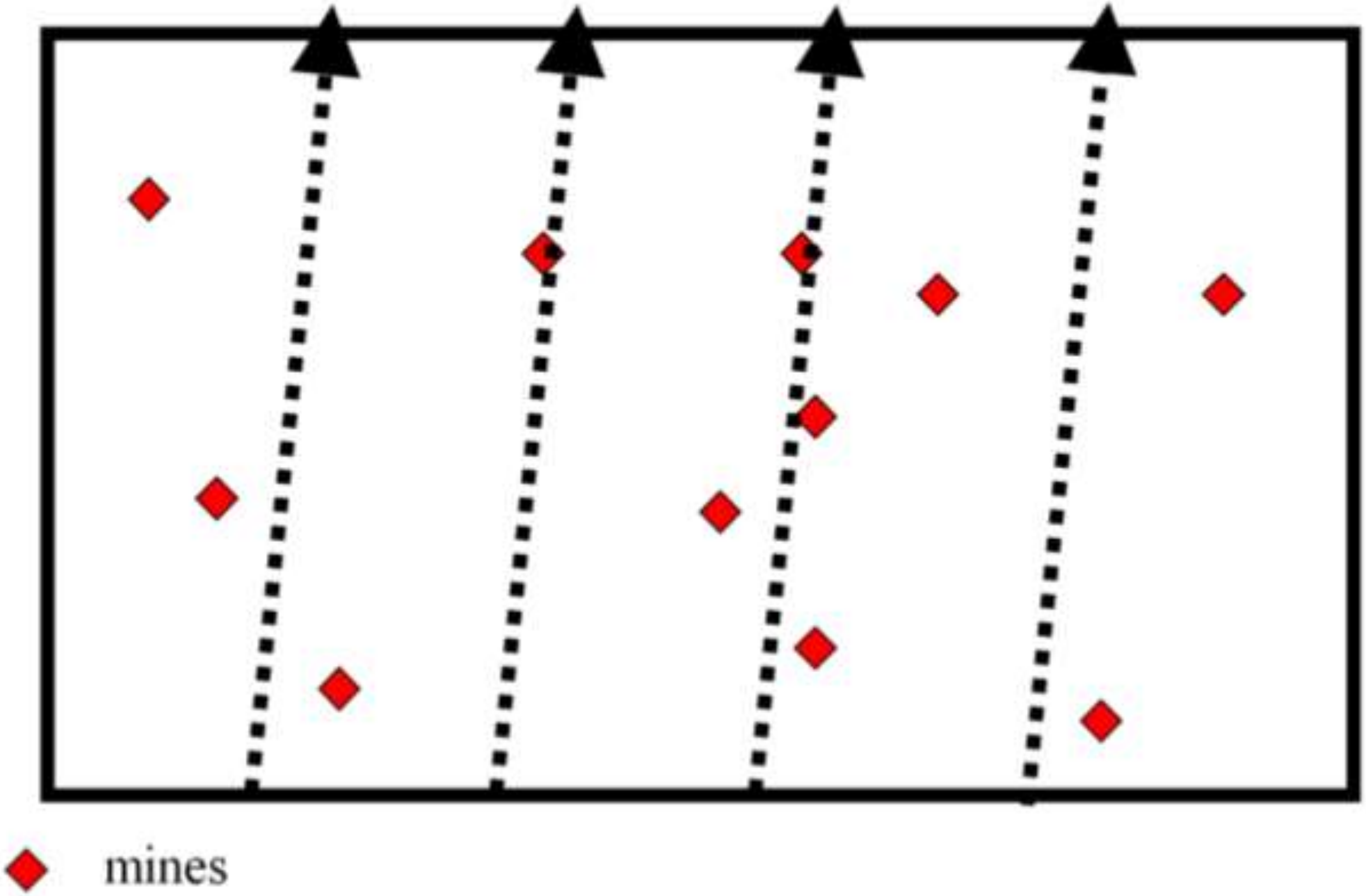
- Which test should we reuse?
- What is the cost of maintaining tests?
  - Complex tests may make extensive use of the environment
  - Complex to maintain
  - Developing test architecture to support tests, e.g., web services
- What is the cost of applying tests?
- What is the cost of applying regression tests?

Fault region model

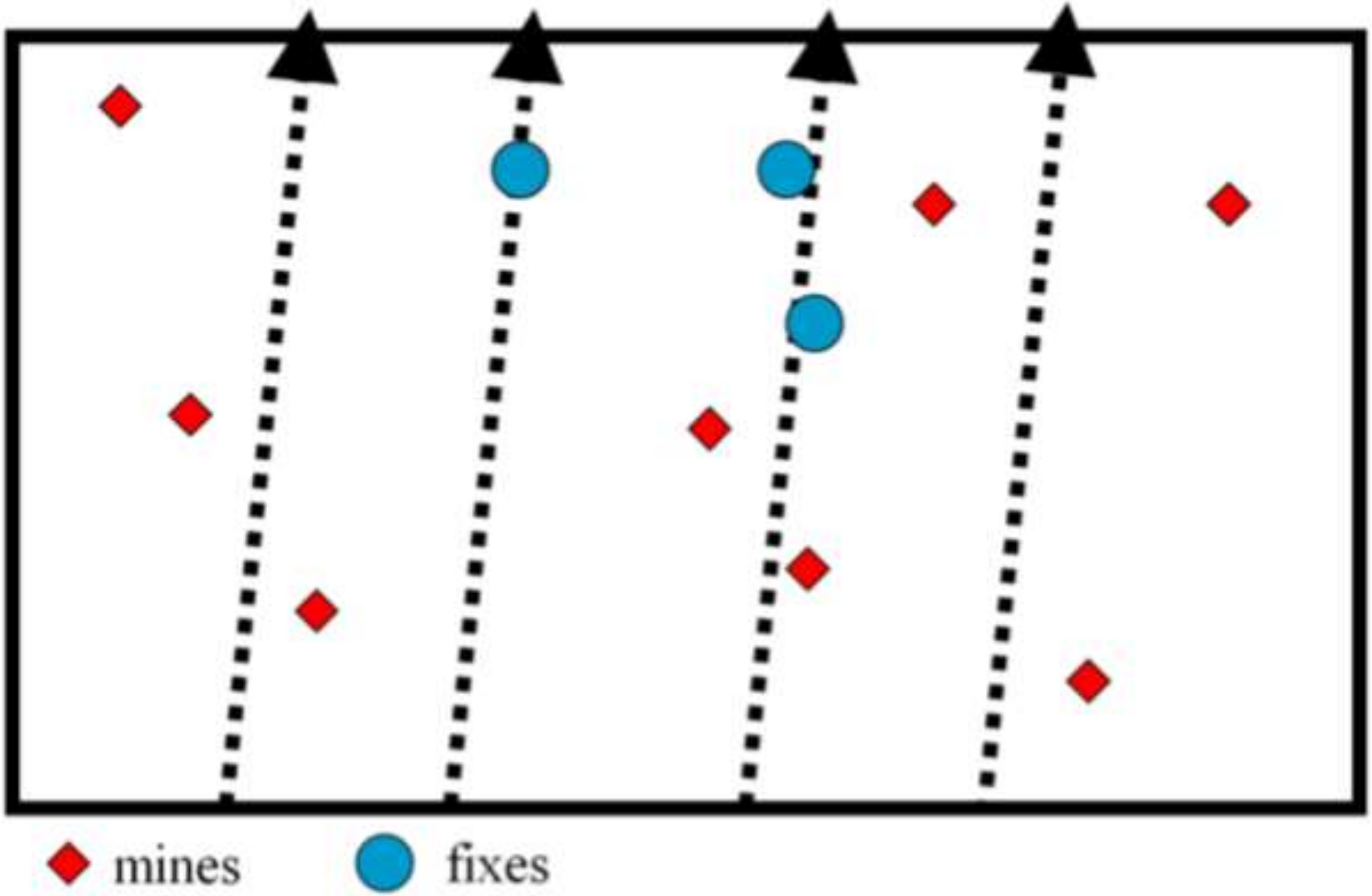
- Systems have fault regions where their behaviour is does not conform to the requirements.
- Tests are point executions of the system.
- Test specifications may specify a region in the input space



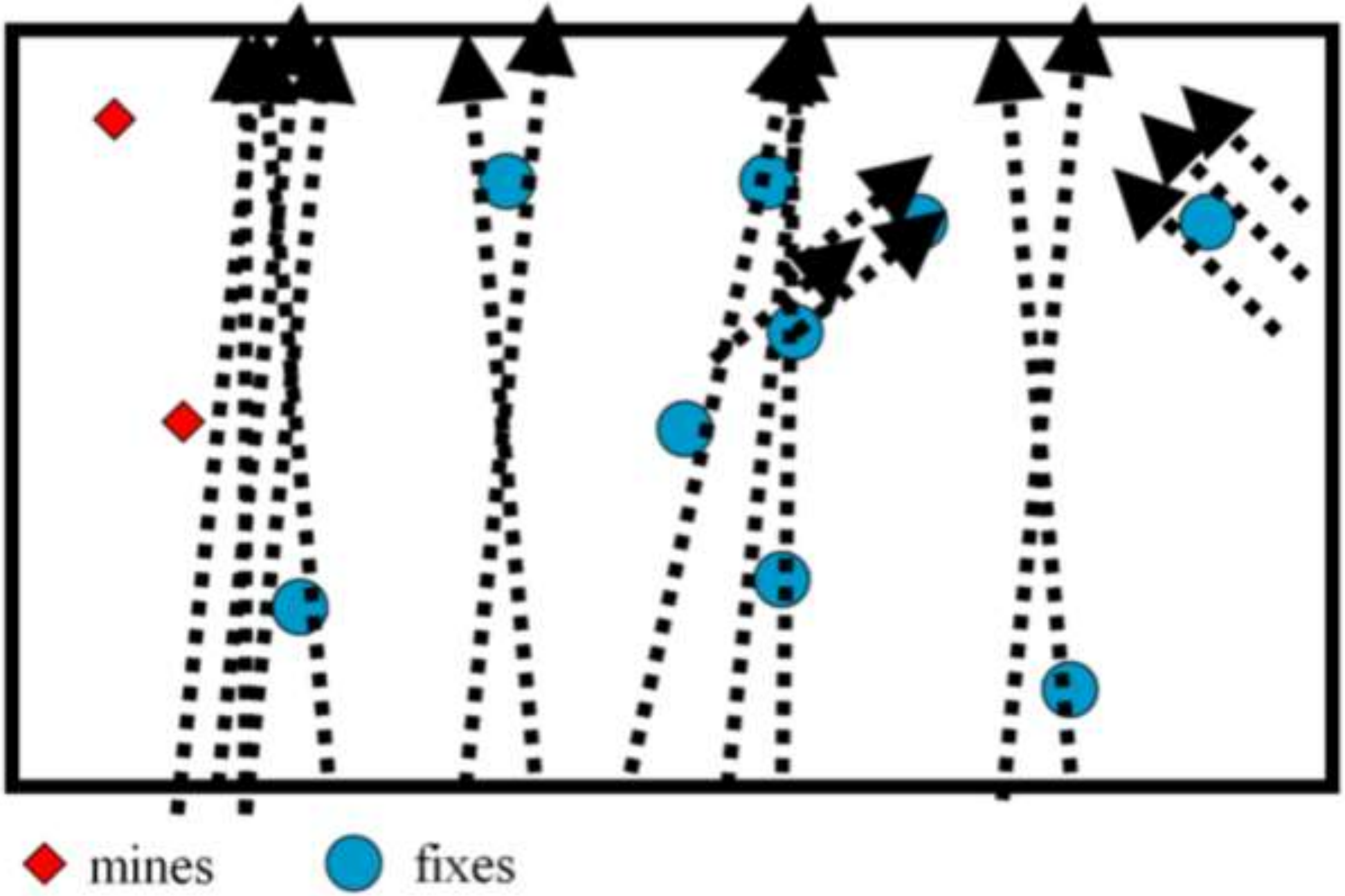
Clearing mines



Totally repeatable tests won’t clear mines



Variable tests are often more effective



Economic perspective

- What is the best way to improve product quality?
  - Maintain a regression test set
  - Develop new tests
  - It is possible to develop new tests for low value events (e.g. patch bundles)
- What is the benefit of reusing tests?
  - Tends to focus on core functionality of the system
  - Perhaps takes a narrow view of the functionality
- Costs:
  - How much does it cost to maintain tests?
  - How much does it cost to create tests?

Support for refactoring

- Tests act as an executable specification.
- Tools like JUnit reduce the cost to the developer.
- Tendency to focus on unit level behaviour.
- Tendency to focus on function over resource use.
- Issues about how to integrate many unit level test sets that have been created individually.

Risk Management

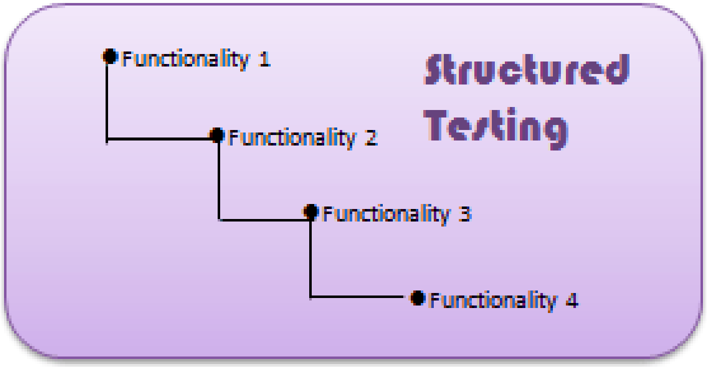
- Tests target critical behaviour the main hazards.
- For embedded systems we have good specifications and it may be possible to infer more from a test result.
- We can use combinations of old tests to exercise the system more extensively on retest:
  - More tests.
  - More combinations of test.
  - More variants.
  - With a good specification we can see how the tests cover the different behaviours of the system.
  - We provide independent testers with a large armoury of possible weapons to break the system.

7.2. Exploratory Testing

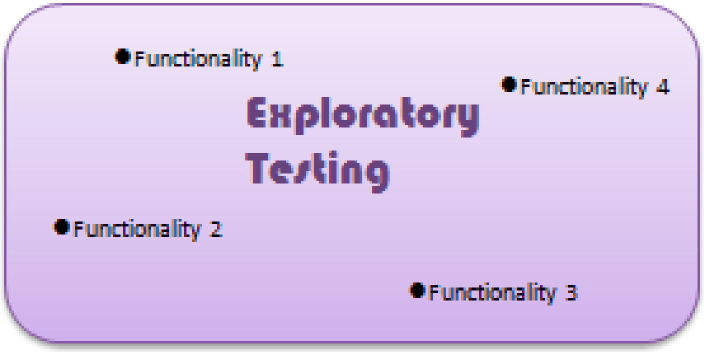
What is Exploratory Testing?

- A style of software testing that
  - **Emphasizes** the **personal freedom** and **responsibility** of the individual tester
  - **Continually optimize the value** of her work
  - Treat test-related learning, test design, test execution, and test results interpretation as **mutually supportive activities** that **run in parallel** throughout the project
- Widely used in Agile models and is all about **discovery, investigation and learning**

Scripted vs Exploratory Testing

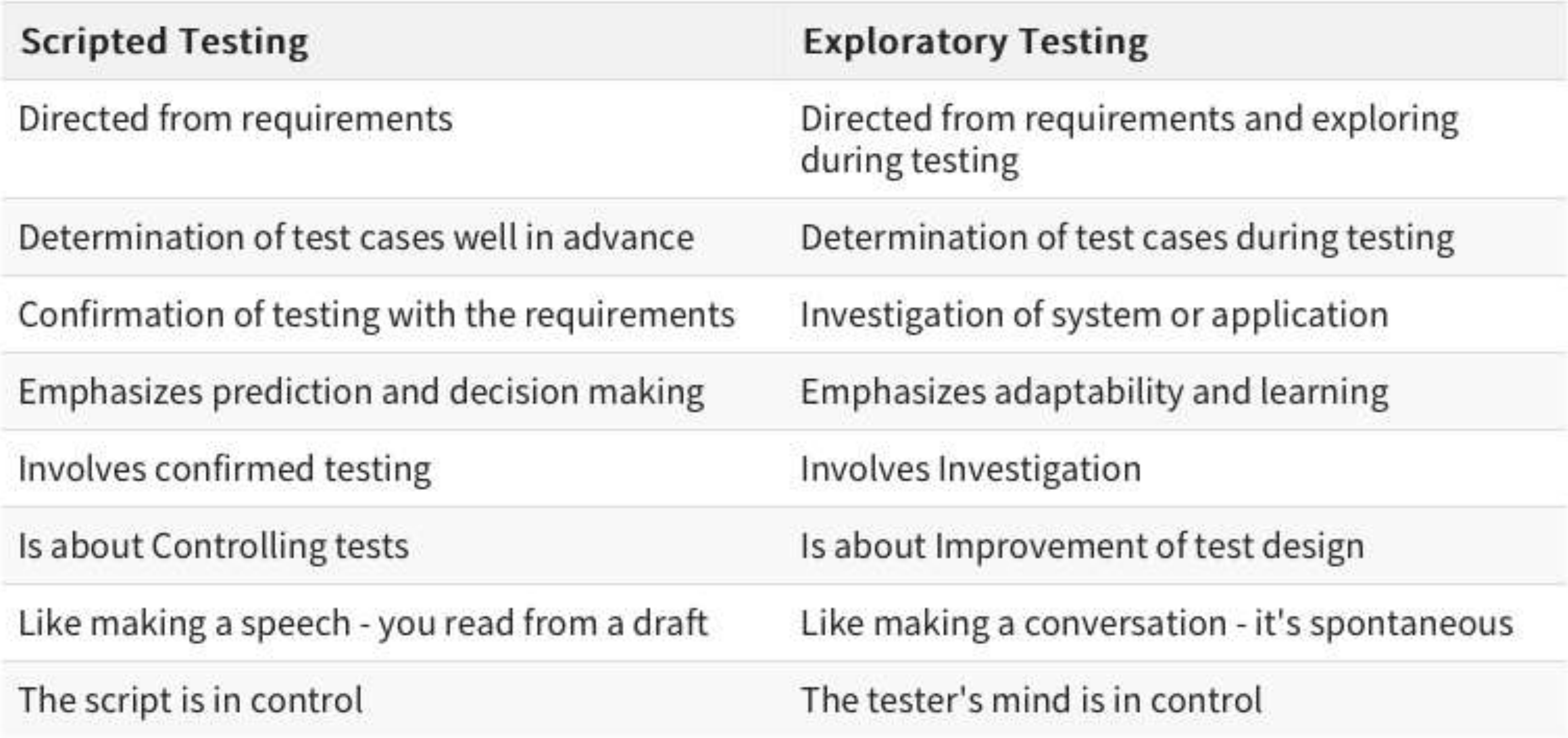


Functionalities are checked in a structured manner



Functionalities are checked in a ad-hoc manner

Scripted vs Exploratory Testing



Scripted Testing	Exploratory Testing
Directed from requirements	Directed from requirements and exploring during testing
Determination of test cases well in advance	Determination of test cases during testing
Confirmation of testing with the requirements	Investigation of system or application
Emphasizes prediction and decision making	Emphasizes adaptability and learning
Involves confirmed testing	Involves Investigation
Is about Controlling tests	Is about Improvement of test design
Like making a speech - you read from a draft	Like making a conversation - it's spontaneous
The script is in control	The tester's mind is in control

Exploratory Testing

- Is not random testing but is ad-hoc testing with a purpose of finding bugs
- Is structured and rigorous
- Is highly teachable and manageable
- Not a technique but an approach

How to do Exploratory Testing?

1. Create a bug taxonomy (classification)
   - Categorize common types of faults found in the past project
   - Analyze the root cause analysis of the problems or faults
   - Find the risks and develop ideas to test the application
2. Test Charter
   - Suggest what to test
   - How it can be tested
   - What need to be looked
   - Help determine how the end user could use the system

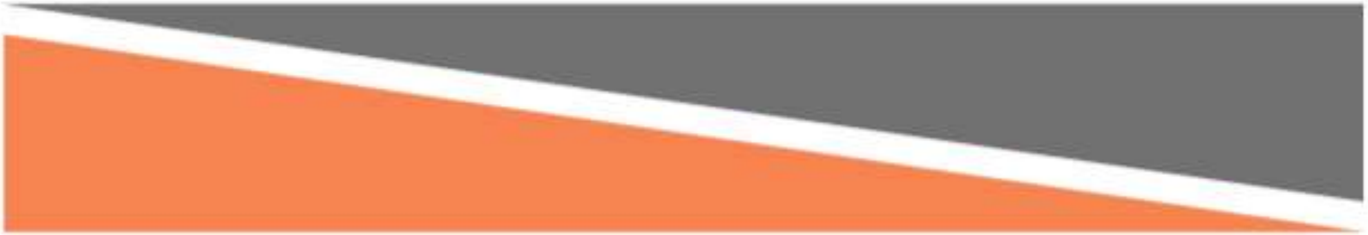
How to do Exploratory Testing?

3. Time Box
   - This method includes a pair of testers working together not less than 90 minutes
   - There should not be any interrupted time in those 90 minutes session
   - Timebox can be extended or reduced by 45 minutes
   - This session encourages testers to react on the response from the system and prepare for the correct outcome

How to do Exploratory Testing?

4. Review Results
   - Evaluation of the defects
   - Learning from the testing
   - Analysis of coverage areas
5. Debriefing
   - Compilation of the output results
   - Compare the results with the charter
   - Check whether any additional testing is needed

Variations of Exploratory Testing



Freestyle

Pure scripted

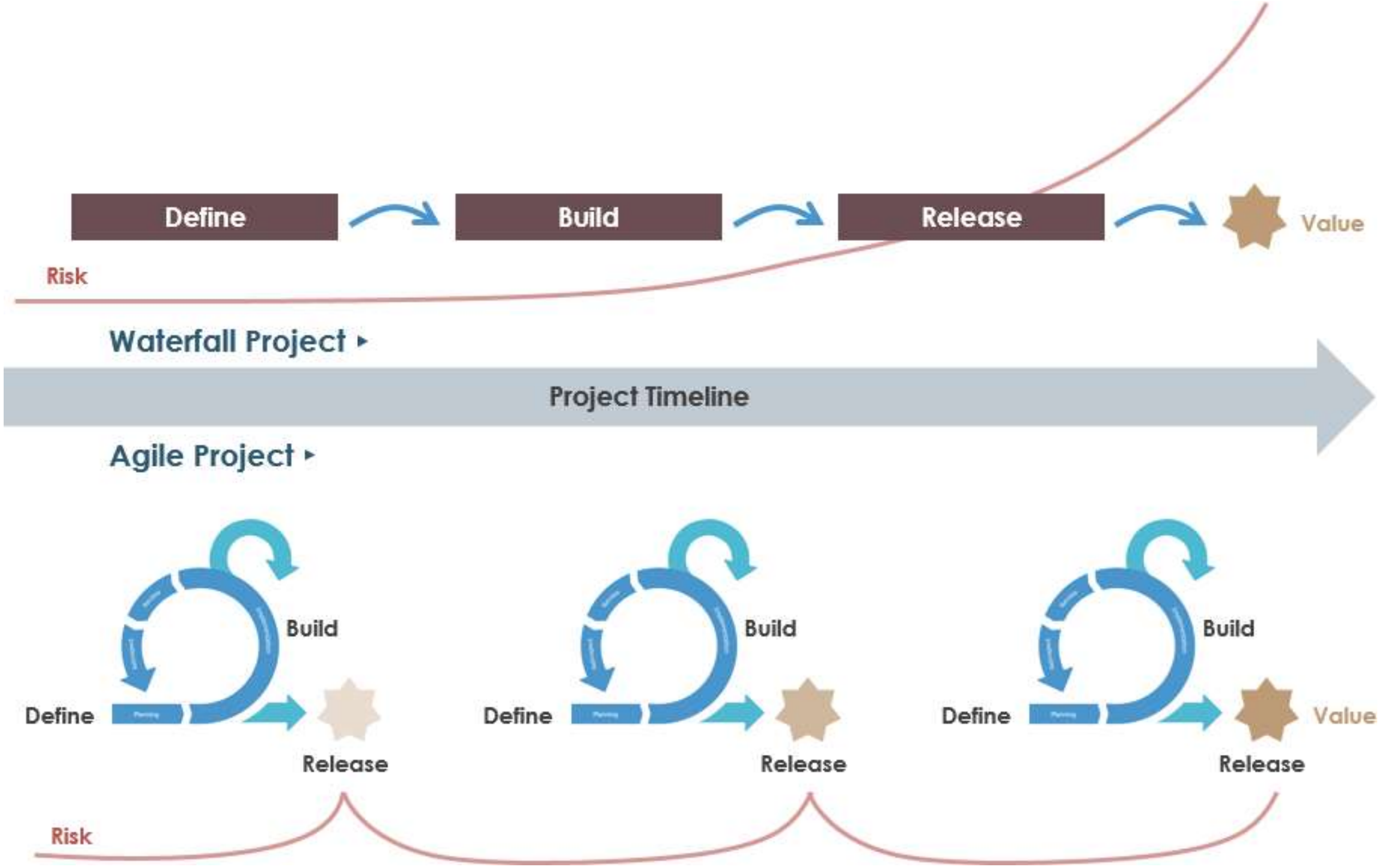
Test object only

Test goals, constraints

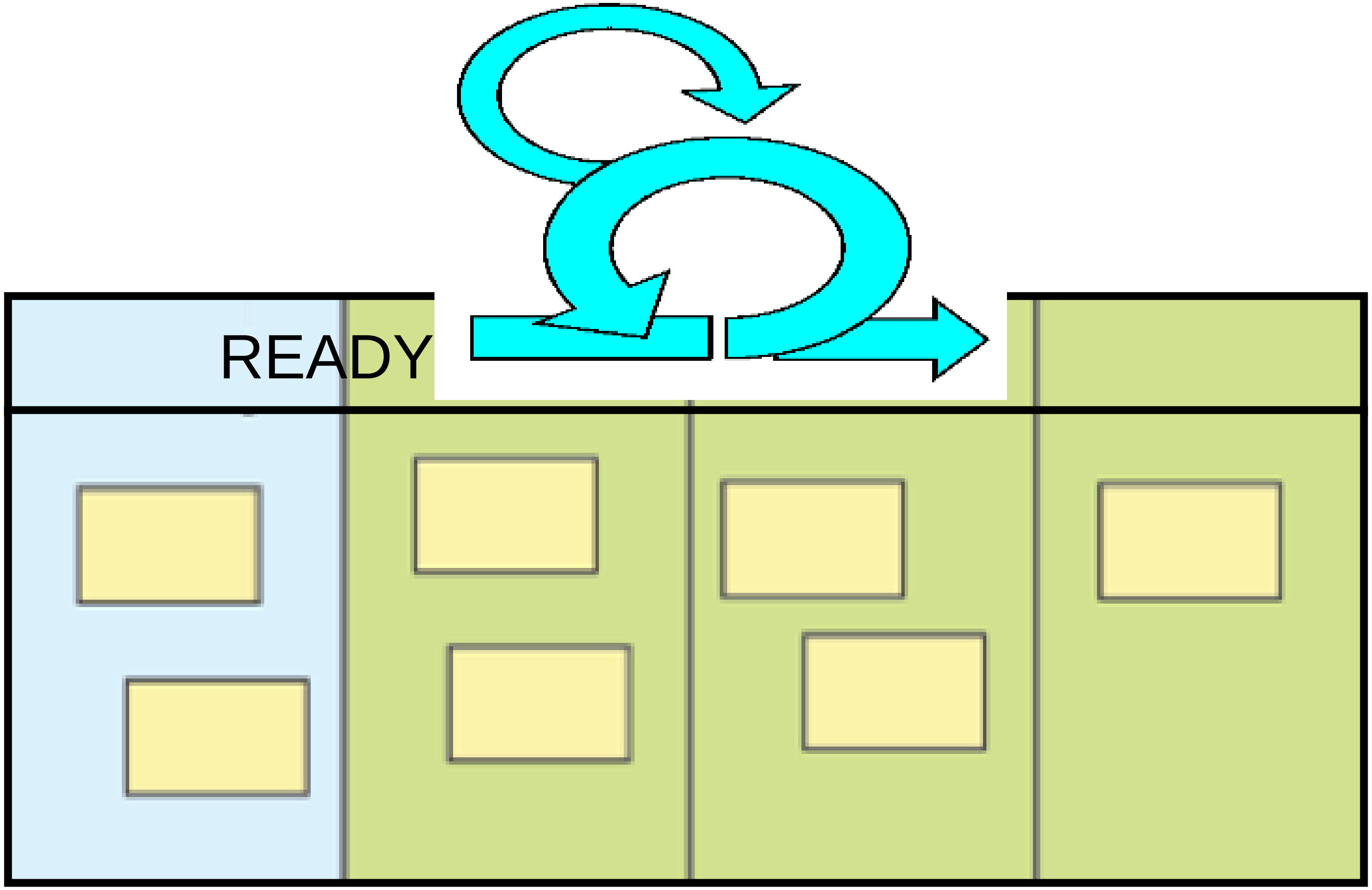
Test object, test steps, test data

7.3. Agile Testing

Software development



Agile software development (*)



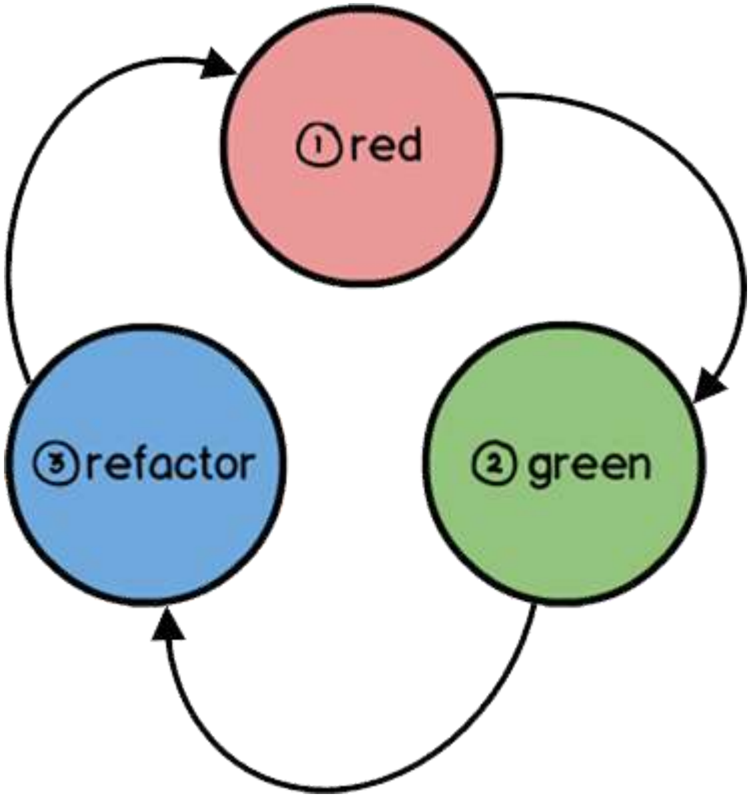
(*) these slides are from: Dave Mattingly

Agile testing

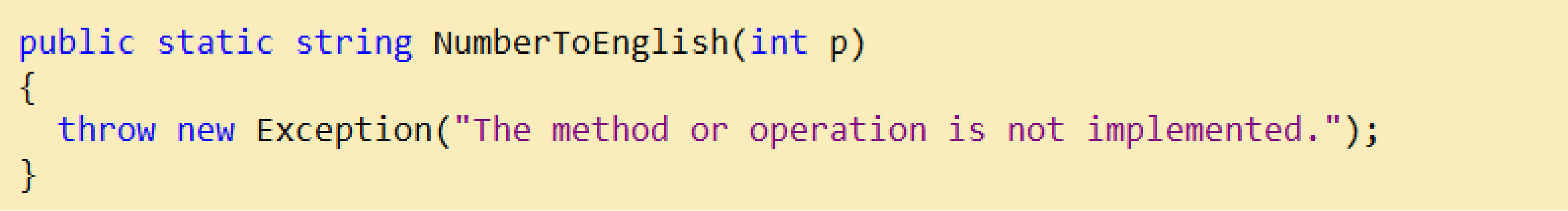
- Imagine, Plan, Make, Test, Deliver

Agile Testing - TDD

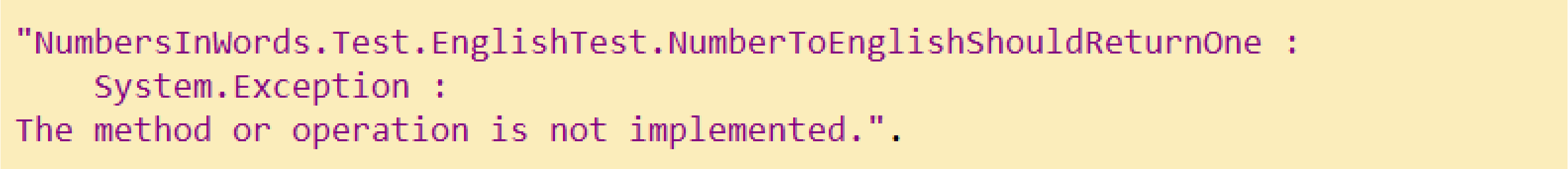
- Test Driven Development
  - Make it Fail
  - Make it Work
  - Make it Better



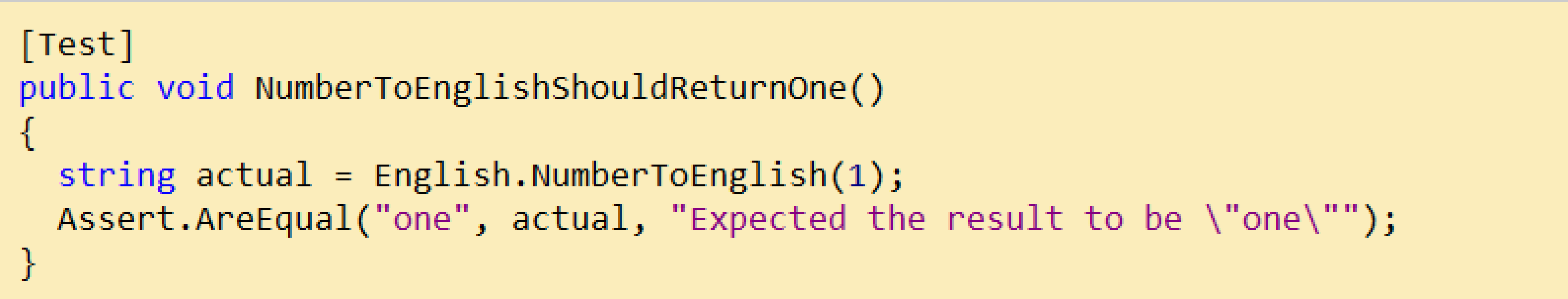
Agile Testing - TDD



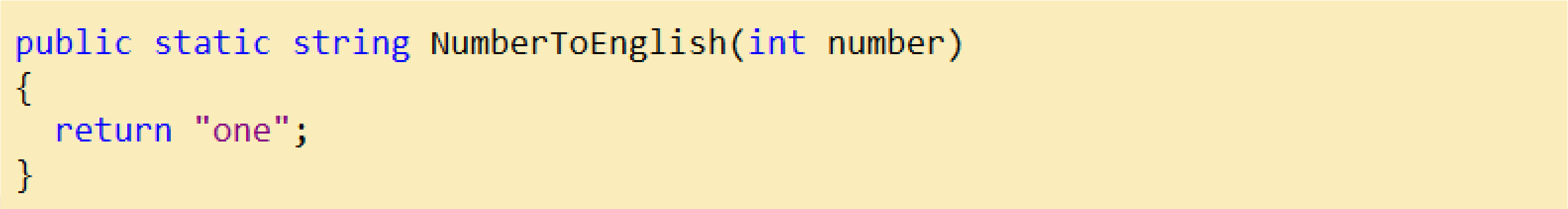
```
public static string NumberToEnglish(int p)
{
  throw new Exception("The method or operation is not implemented.");
}
```



```
"NumbersInWords.Test.EnglishTest.NumberToEnglishShouldReturnOne :
    System.Exception :
The method or operation is not implemented.".
```



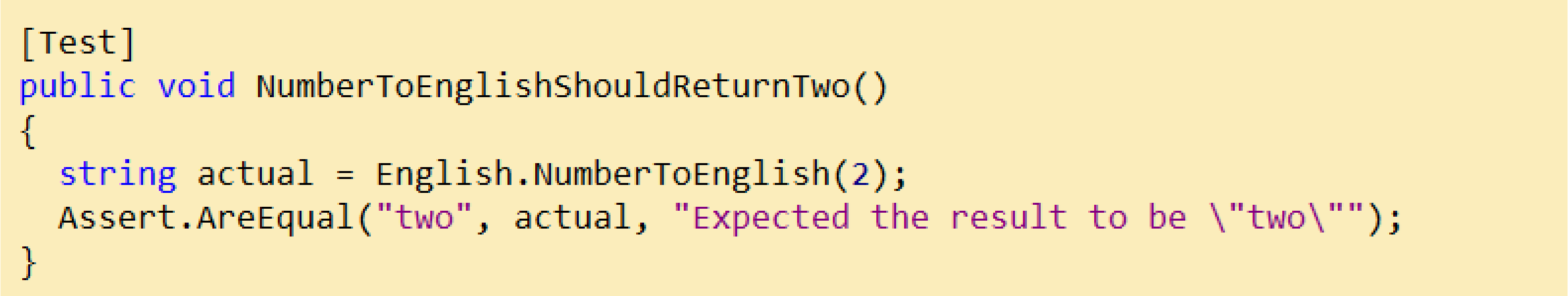
```
[Test]
public void NumberToEnglishShouldReturnOne()
{
  string actual = English.NumberToEnglish(1);
  Assert.AreEqual("one", actual, "Expected the result to be \"one\"");
}
```



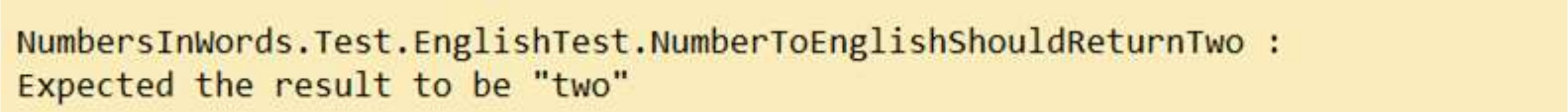
```
public static string NumberToEnglish(int number)
{
  return "one";
}
```

Agile Testing - TDD

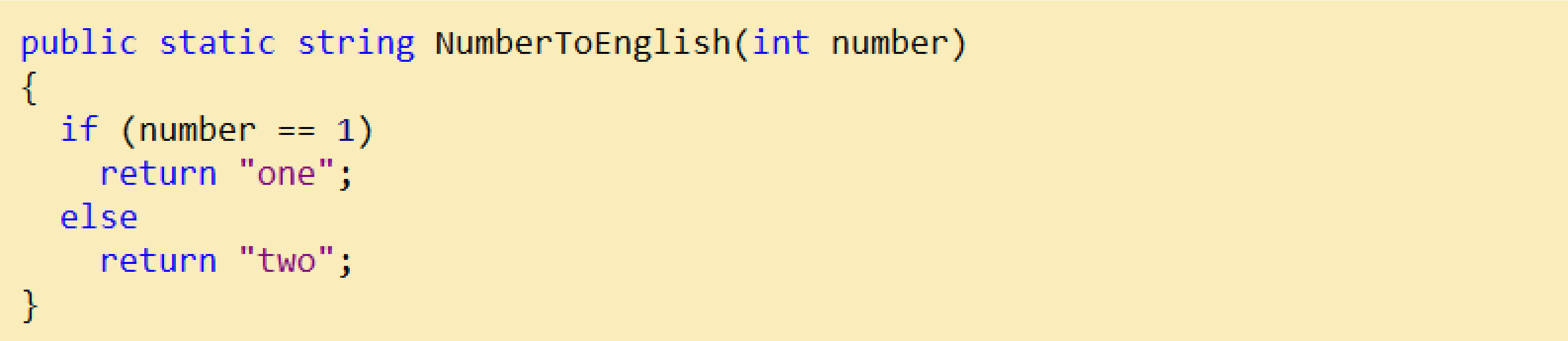
- Tools: csUnit, jUnit, nUnit, BusterJS



```
[Test]
public void NumberToEnglishShouldReturnTwo()
{
  string actual = English.NumberToEnglish(2);
  Assert.AreEqual("two", actual, "Expected the result to be \"two\"");
}
```



```
NumbersInWords.Test.EnglishTest.NumberToEnglishShouldReturnTwo :
Expected the result to be "two"
```

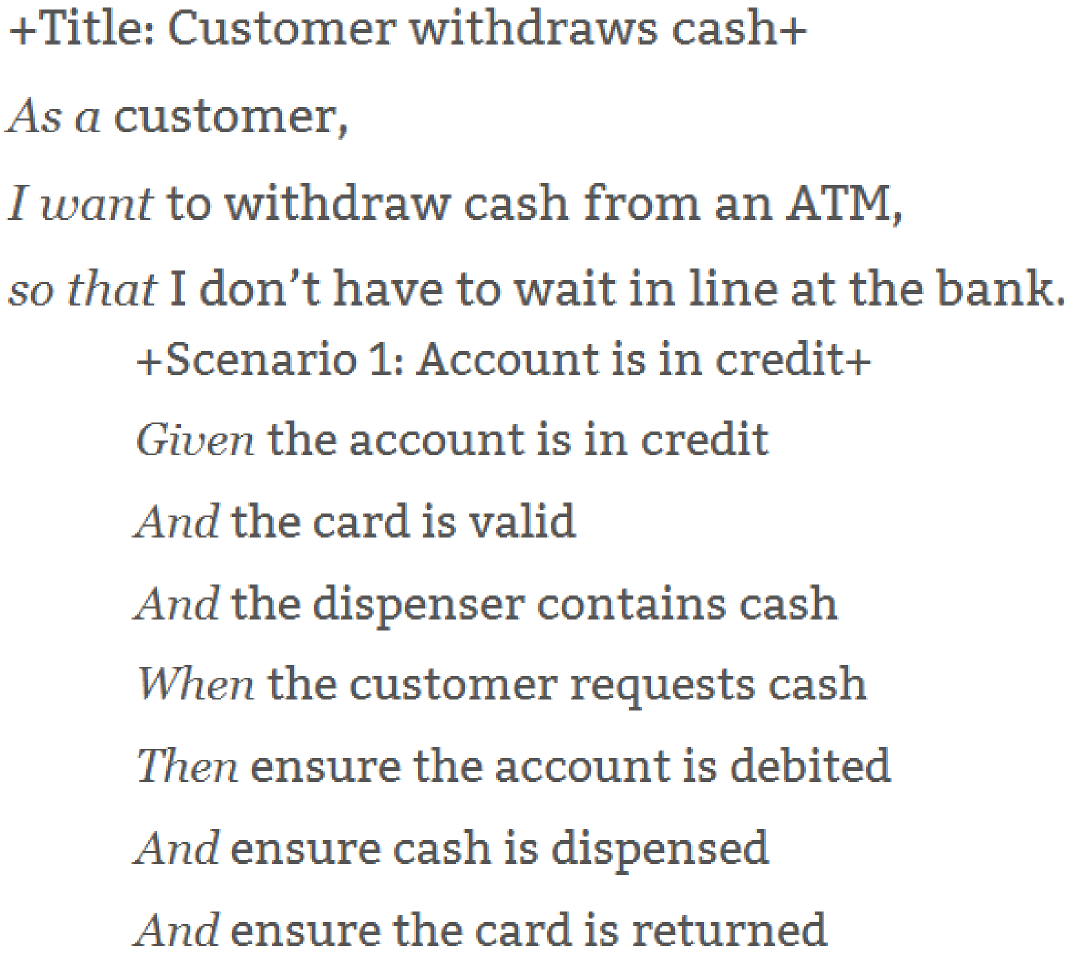


```
public static string NumberToEnglish(int number)
{
  if (number == 1)
    return "one";
  else
    return "two";
}
```

Agile testing - BDD

- Behavior Driven Development
  - Given
  - When
  - Then

Agile testing - BDD



+Title: Customer withdraws cash+

As a customer,

I want to withdraw cash from an ATM,

so that I don’t have to wait in line at the bank.

+Scenario 1: Account is in credit+

Given the account is in credit

And the card is valid

And the dispenser contains cash

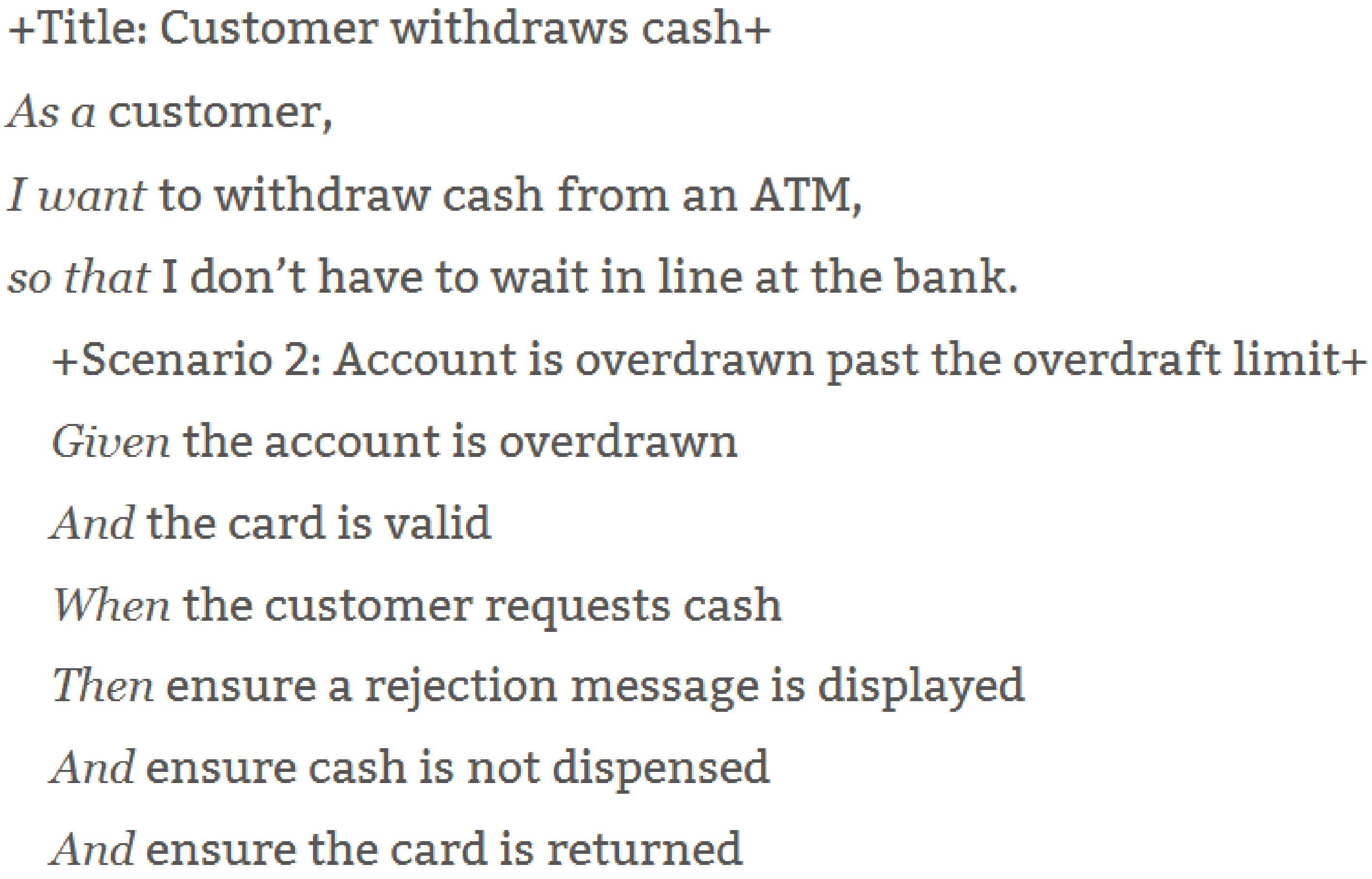
When the customer requests cash

Then ensure the account is debited

And ensure cash is dispensed

And ensure the card is returned

Agile testing - BDD



+Title: Customer withdraws cash+

As a customer,

I want to withdraw cash from an ATM,

so that I don't have to wait in line at the bank.

+Scenario 2: Account is overdrawn past the overdraft limit+

Given the account is overdrawn

And the card is valid

When the customer requests cash

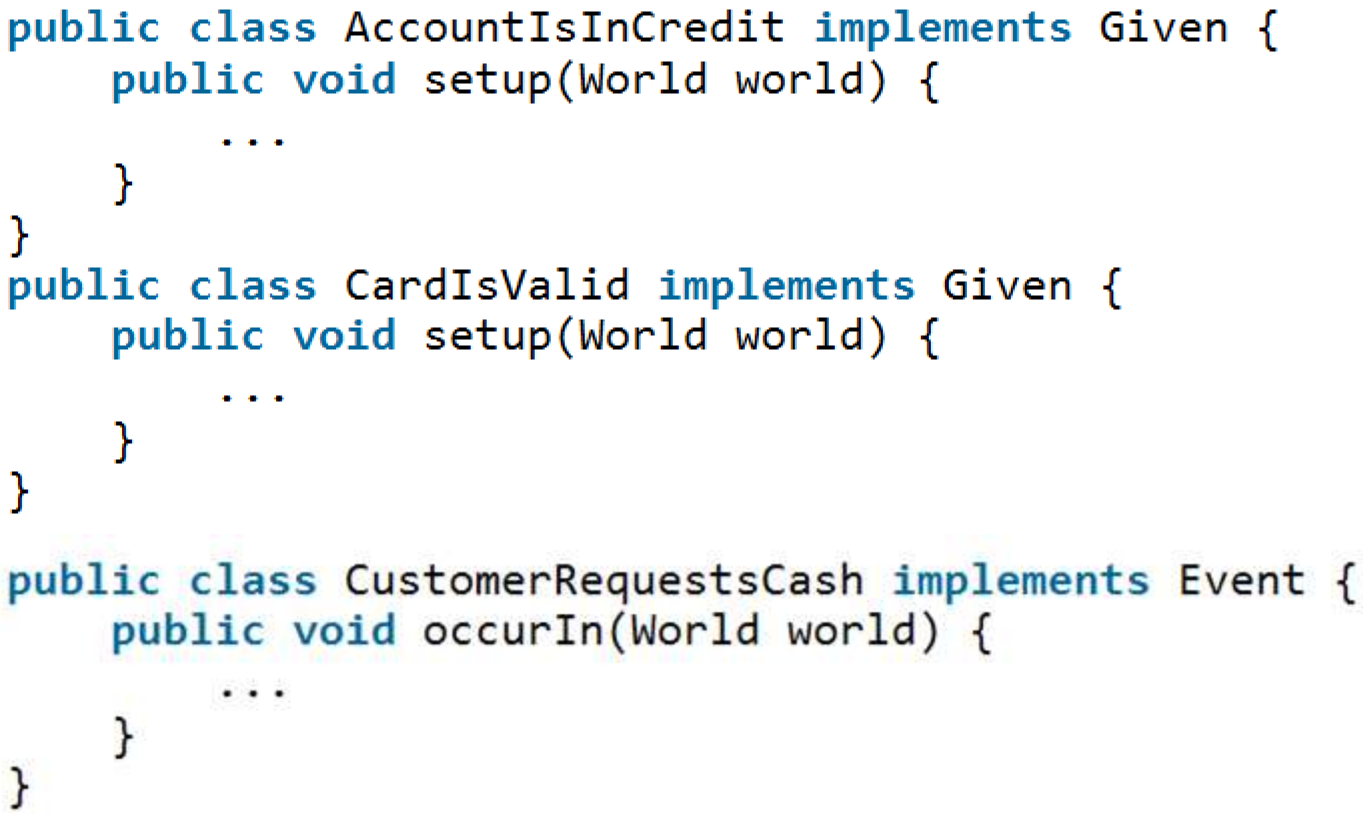
Then ensure a rejection message is displayed

And ensure cash is not dispensed

And ensure the card is returned

Agile testing - BDD

- Tools: Cucumber, RSpec, SpecFlow

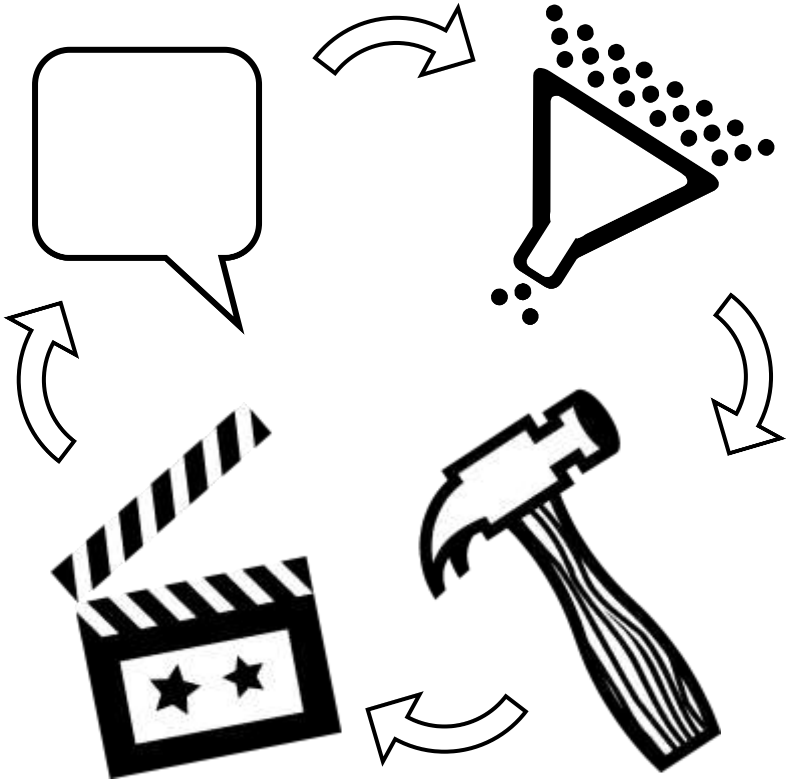


```
public class AccountIsInCredit implements Given {
    public void setup(World world) {
        ...
    }
}
public class CardIsValid implements Given {
    public void setup(World world) {
        ...
    }
}

public class CustomerRequestsCash implements Event {
    public void occurIn(World world) {
        ...
    }
}
```

Agile testing - ATDD

- Acceptance Test Driven Development
  - Discuss
  - Distill
  - Develop
  - Demonstrate

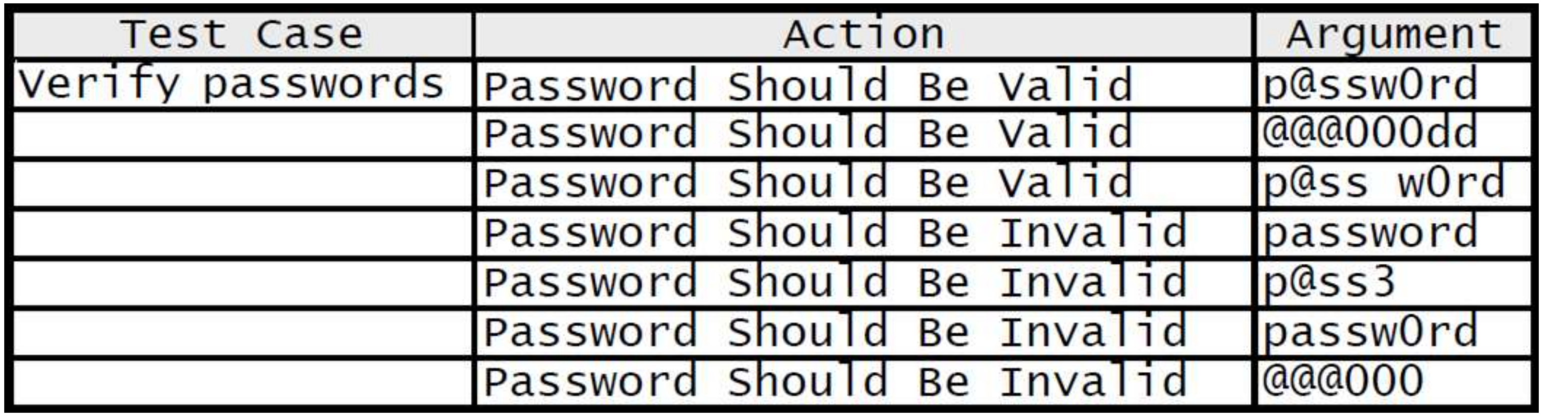


Agile testing - ATDD

- Discuss
  - What is a valid password?
  - What characters are mandatory?
  - When should they change?
  - Can changed passwords repeat?
  - How will we know it works?
  - What are some specific examples?

Agile testing - ATDD

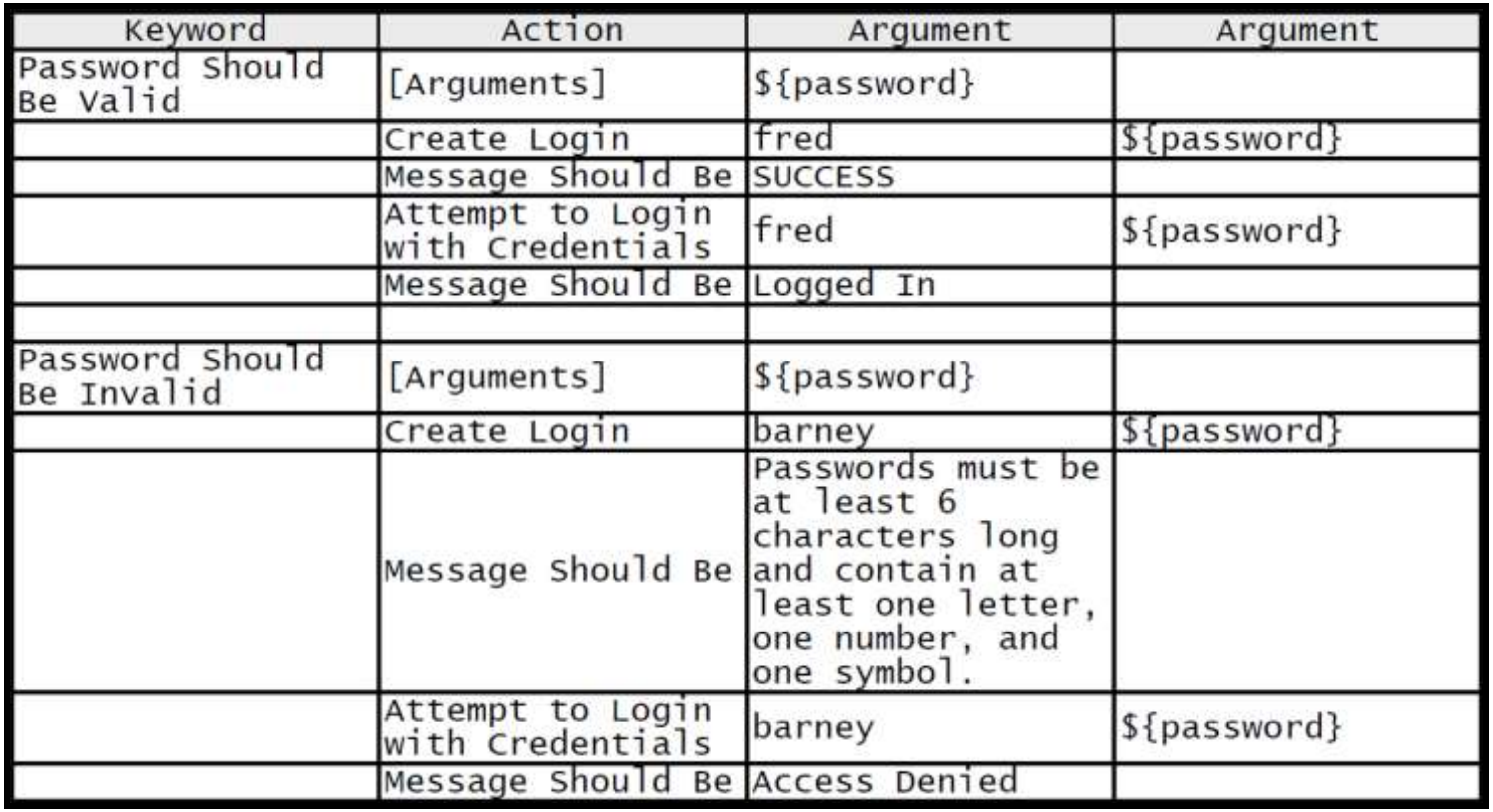
- Distill



Test Case	Action	Argument
Verify passwords	Password Should Be Valid	p@ssw0rd
	Password Should Be Valid	@@@000dd
	Password Should Be Valid	p@ss w0rd
	Password Should Be Invalid	password
	Password Should Be Invalid	p@ss3
	Password Should Be Invalid	passw0rd
	Password Should Be Invalid	@@@000

Agile testing - ATDD

- Develop



Keyword	Action	Argument	Argument
Password Should Be Valid	[Arguments]	${password}	
	Create Login	fred	${password}
	Message Should Be	SUCCESS	
	Attempt to Login with Credentials	fred	${password}
	Message Should Be	Logged In	
Password Should Be Invalid	[Arguments]	${password}	
	Create Login	barney	${password}
	Message Should Be	Passwords must be at least 6 characters long and contain at least one letter, one number, and one symbol.	
	Attempt to Login with Credentials	barney	${password}
	Message Should Be	Access Denied	

Agile testing - ATDD

- Demonstrate

- Tools: EasyB, FitNesse, JBehave, SpecTacular

Agile Testing - Auto

- Automated Regression Testing
  - Simulates real-world experiences
  - Eliminates repetitive tests
  - Eases complex tests

Agile Testing - Auto

- Tools: Selenium, Silk, Concordion

Considerations

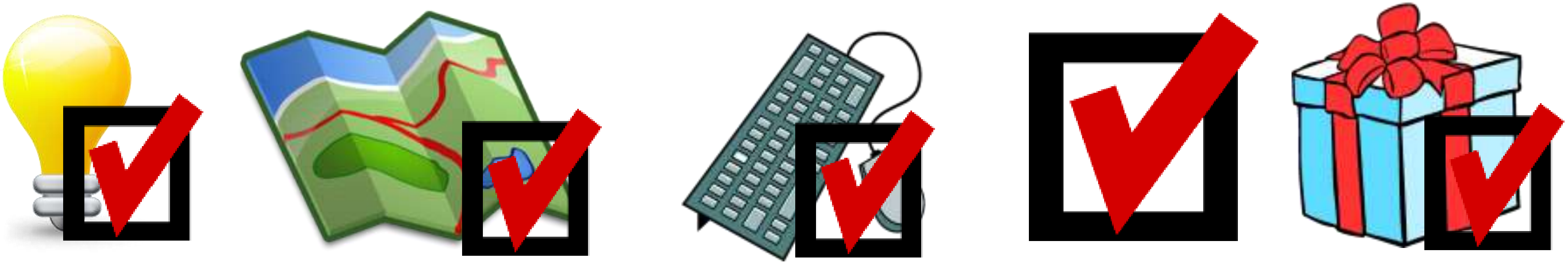
- TDD – implementation
  - Is it working?
- BDD – system behavior
  - Is it rights?
- ATDD – requirements
  - Is it useful?
- Automated Regression – availability
  - Is it reliable?

Considerations

- Adoption
- Promotion
- Bugs
- Documentation
- Versioning
- Notifications

Considerations

- Test everywhere

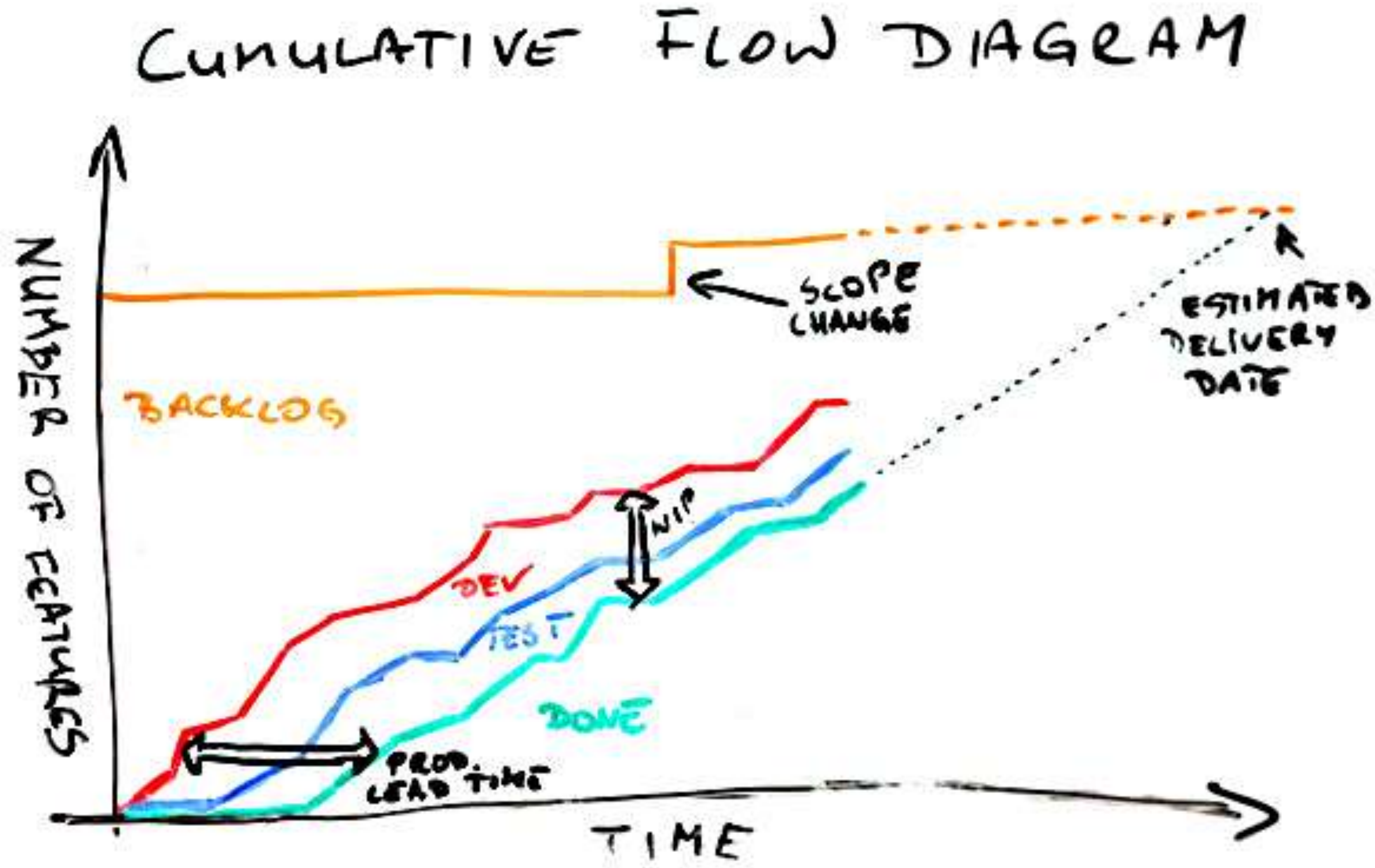


ATDD BDD TDD QA Auto

Considerations

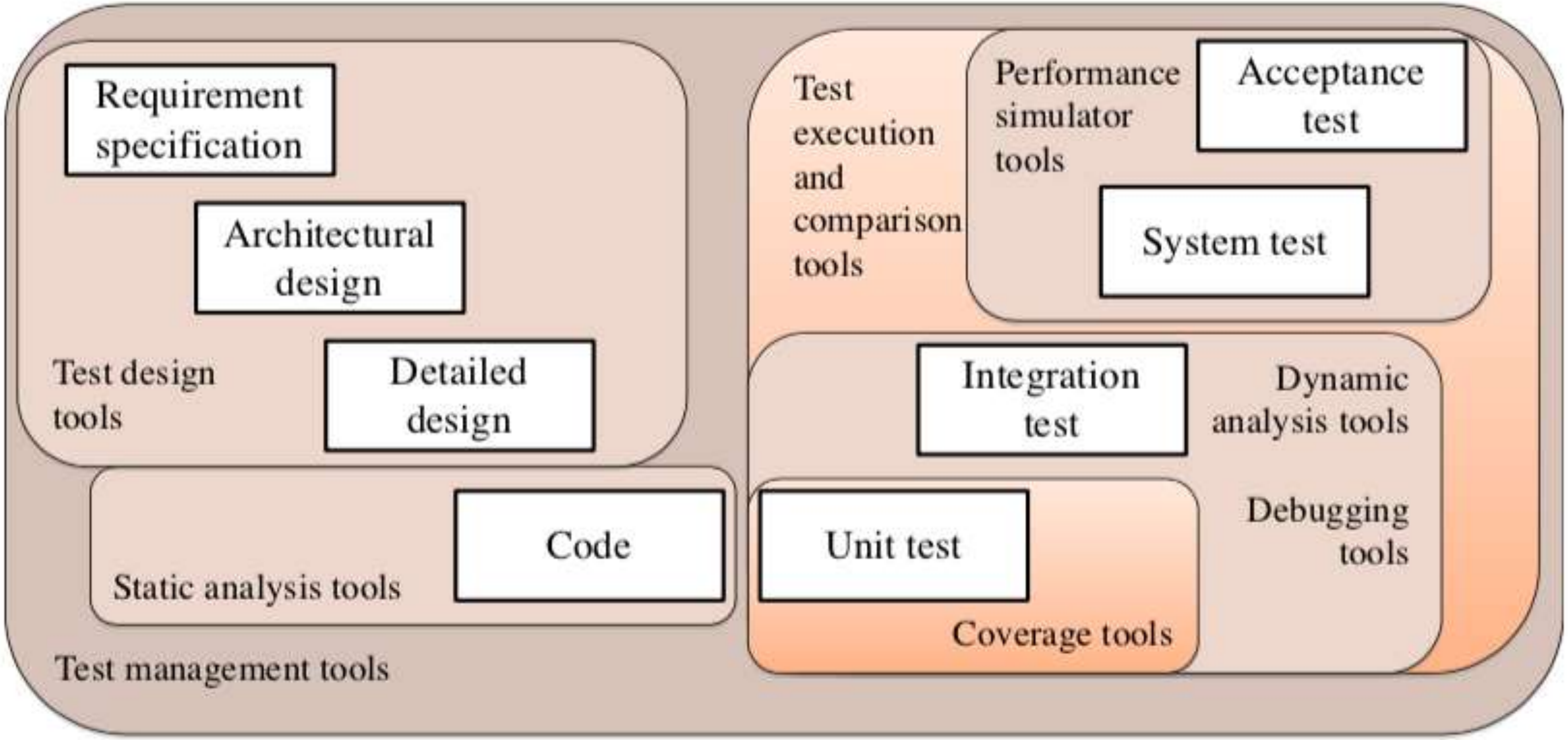
- Applications
- Data
- Performance
- Availability
- Roles
- Accessibility
- Security

Considerations



7.4. Automation Testing

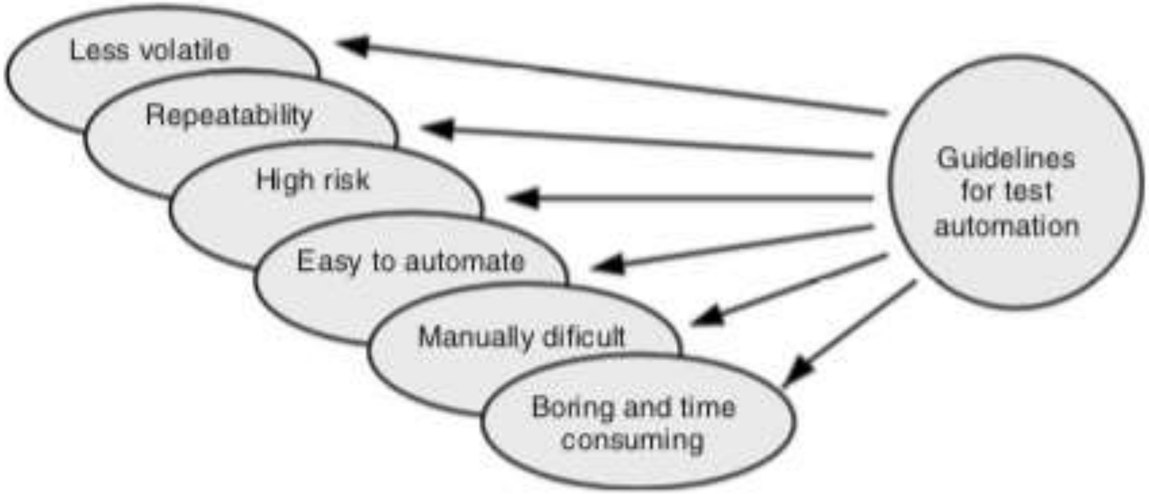
Testing tools by process



What to automate?

Test cases that are:

- Less volatile
- Repeatable
- High risk
- Easy to automate
- Manually difficult
- Boring and time consuming

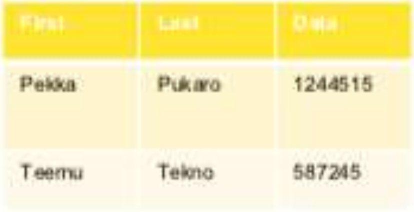


Evolution of Test Automation

1. Recorded Scripts

2. Engineered Scripts

3. Data-driven Testing

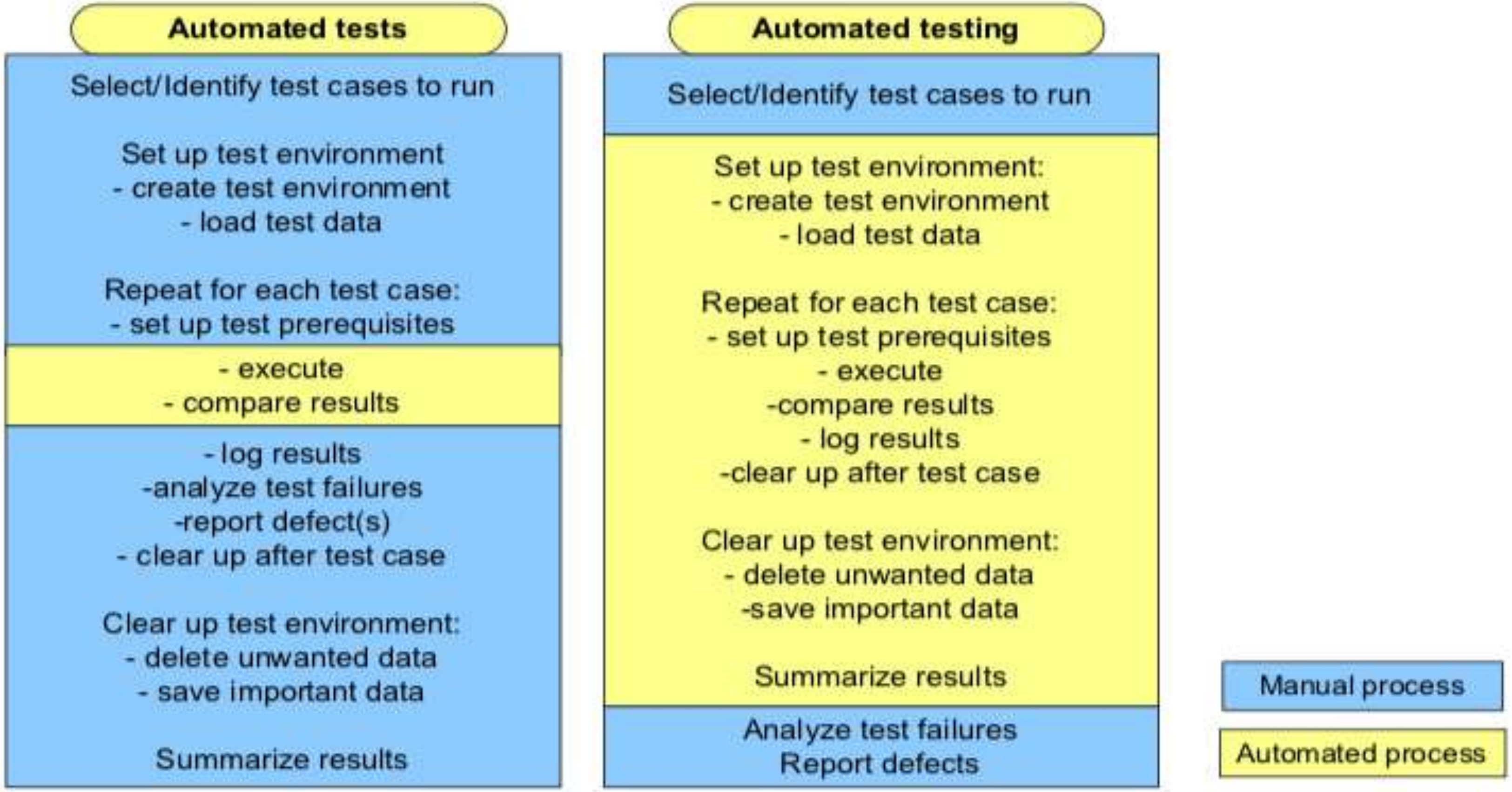


First	Last	Data
Pekka	Pukaro	1244515
Teemu	Tekno	587245

4. Keyword-driven Testing

5. Model-based Testing

Automating different steps



Automated tests

Select/Identify test cases to run

Set up test environment
- create test environment
- load test data

Repeat for each test case:
- set up test prerequisites
- execute
- compare results
- log results
-analyze test failures
-report defect(s)
- clear up after test case

Clear up test environment:
- delete unwanted data
- save important data

Summarize results

Automated testing

Select/Identify test cases to run

Set up test environment:
- create test environment
- load test data

Repeat for each test case:
- set up test prerequisites
- execute
-compare results
- log results
-clear up after test case

Clear up test environment:
- delete unwanted data
-save important data

Summarize results

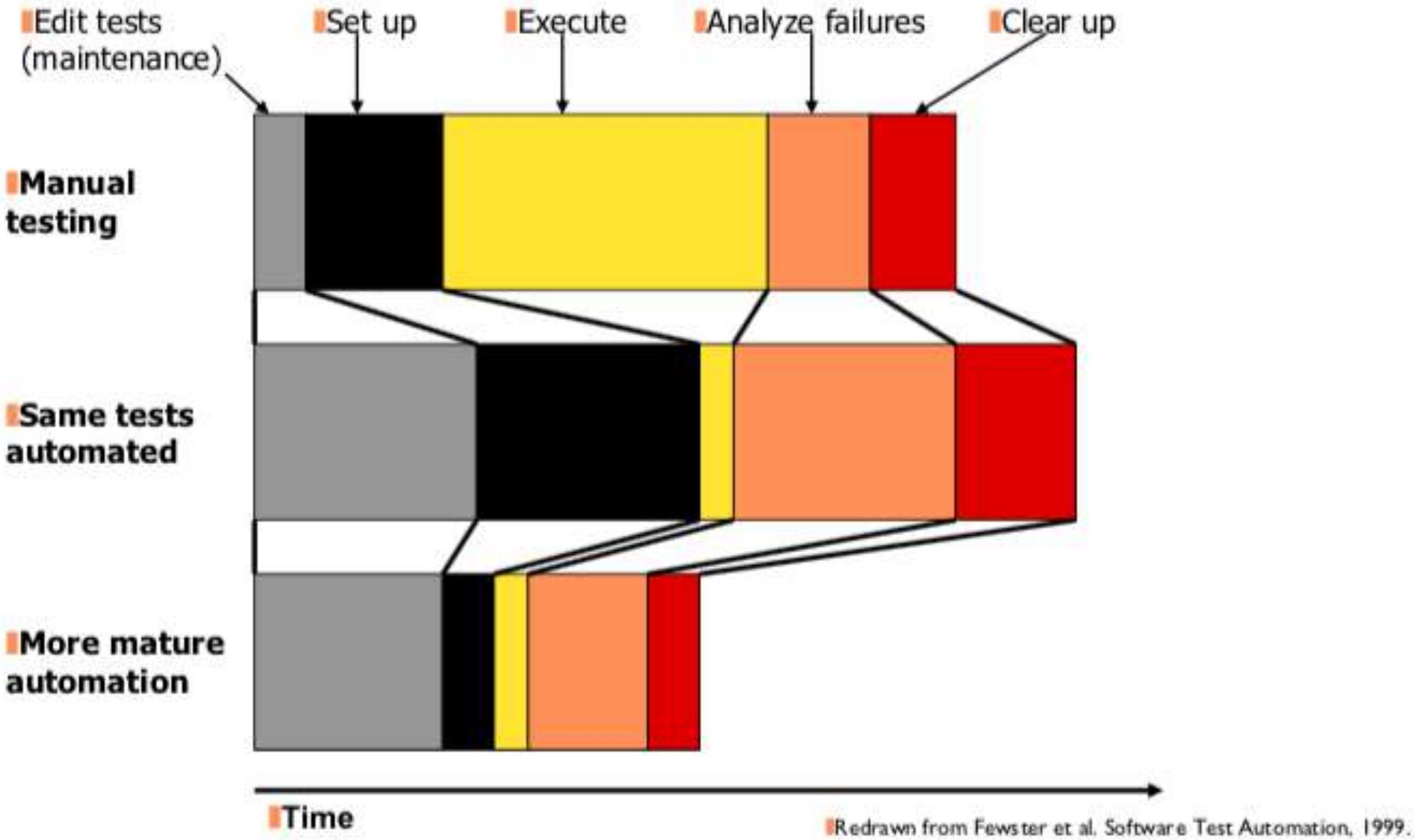
Analyze test failures
Report defects

Manual process

Automated process

Redrawn from Fewster et al. Software Test Automation, 1999.

Relationship of testing activities



Redrawn from Fewster et al. Software Test Automation, 1999.

Test automation promises

1. Efficient regression test
2. Run tests more often
3. Perform difficult tests (e.g. load, outcome check)
4. Better use of resources
5. Consistency and repeatability
6. Reuse of tests
7. Earlier time to market
8. Increased confidence

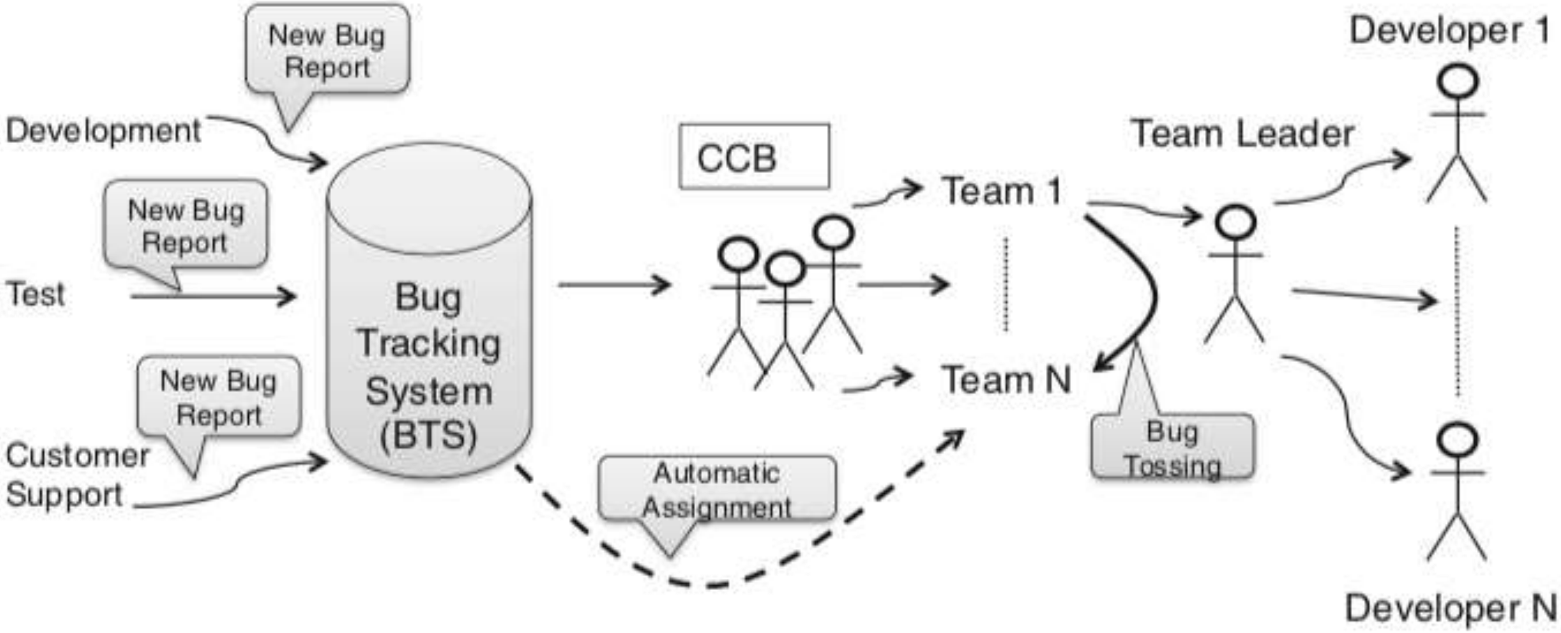
Common problems

1. Unrealistic expectations
2. Poor testing practice
3. Expected effectiveness
4. False sense of security
5. Maintenance of automatic tests
6. Technical problems (e.g. Interoperability)
7. Organizational issues

Limits of automated testing

- Does not replace manual testing
- Manual tests find more defects than automated tests
  - Does not improve effectiveness
- Greater reliance on quality of tests
  - Oracle problem
- Test automation may limit the software development
  - Costs of maintaining automated tests

New automation perspectives



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attention!!!