



STEVENS
INSTITUTE *of* TECHNOLOGY
THE INNOVATION UNIVERSITY®

SSW-555: Agile Methods for Software Development

Testing and Continuous Integration

Prof. Zhongyuan Yu
School of Systems and Enterprises
Stevens Institute of Technology





Today's Topics

- Overview of testing
 - Definition
 - Testing stages
- Comparing testing in traditional plan driven approach and agile approach
- Test-First (or Test-Driven) Development
- From Test-Driven to Behavior-Driven

Acknowledgements

- <https://www.cs.tau.ac.il/~nachumd/horror.html>
- <https://www.guru99.com/software-testing-introduction-importance.html>
- <https://www.tutorialspoint.com/junit/>
- <http://junit.org/junit4/javadoc/4.12/org/junit/package-summary.html>
- <https://docs.python.org/dev/library/unittest.html>
- <http://www.softwaretestinggenius.com/all-about-system-testing-an-important-part-of-our-software-testing-effort>
- <http://softwaretestingfundamentals.com/acceptance-testing/>

What is Software Testing?

The process of *dynamically exercising a program*, to determine whether a software system meets specific quality requirements such as conformance to specifications, performance, reliability, or robustness, etc.

- Does the system do what it's supposed to do?



Why is Testing Important?

Software failures can cause monetary and human loss...



<https://www.cs.tau.ac.il/~nachumd/horror.html>

<https://www.guru99.com/software-testing-introduction-importance.html>

Air Crash

- A China Airlines Airbus A300 crashes on April 26, 1994



264 people killed

Therac-25 Failure

- The Therac-25 radiation overdose due to software bugs



4 patients killed, 2 life-long injured

The Most Expensive Hyphen

- 1962, the Mariner I exploded less than 5 minutes into flight due to a tiny typo in mathematical code.



*\$ 80 million went to waste in 1962
≈ \$724 million in 2021*

U.S. Bank Accounts

- May 1996, the bank updated its ATM transaction software with new message codes. The message codes were unfortunately not tested on all ATM protocols



823 customers paid \$920 million

Software Testing is Important!

Software bugs could be expensive and dangerous!



Software Testing Methods



Black box testing: Examines the *functionality* of an application without peering into its internal structures or workings.



White box testing: tests **internal structures** or workings of an application, as opposed to its functionality.



Software Testing Phases

1. Unit Test: Test new features
2. Integration Test: Combine and test code of multiple developers
3. Regression Test: Verify that new changes haven't broken previous working code
4. System Test: Test the entire system as a whole
 - Functionality, performance, stress, security
5. User Acceptance Test: Verify that the customers are happy!




Unit Testing (xUnit)

- Framework for unit tests, e.g. JUnit, PyUnit(unittest)
- Tests are written as class methods
 - Developers appreciate that tests are code
- Test code is separate from production code
- There are similar xUnit frameworks and tools for many other programming languages
 - CUnit
 - CppUnit
 - csUnit
 - ...



Architecture of a Test

1. **fixture** –  creates objects and context for test
2. **exercise** – invokes methods under test
3. **result** – asserts equality (or something else) about the results of the exercise

Assert equal, not equal, close, membership, etc.

Simple Python unittest



```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

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


Simple Python Unittest



`import unittest` 1. Import unittest module that provides classes for supporting tests

```
class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

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

Simple Python unittest



```
import unittest
```

```
class TestStringMethods(unittest.TestCase):
```

2. Derive a test class from *unittest.TestCase*. The test class name is arbitrary

```
    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')
```

```
    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())
```

```
    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)
```

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

Simple Python unittest



```
import unittest
```

```
class TestStringMethods(unittest.TestCase):
```

```
    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')
```

```
    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())
```

```
    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)
```

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

3. Define a set of methods for each Test case.

- Each test case contains calls to `unittest.TestCase.assert*()`
- Method name should begin with `'test_'`

Simple Python unittest



```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

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

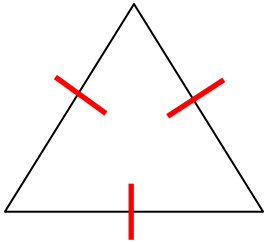
4. Call `unittest.main()` to automatically invokes all methods in test class



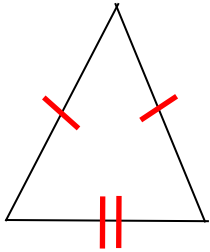
Python unittest Assert Methods

Method	Checks
<code>assertEqual(a, b, msg=None)</code>	<code>a == b</code>
<code>assertNotEqual(a, b, msg=None)</code>	<code>a != b</code>
<code>assertAlmostEqual(a, b, places=7, msg=None)</code>	<code>round(a-b, places) == 0</code>
<code>assertNotAlmostEqual(a, b, places=7, msg=None)</code>	<code>round(a-b, places) != 0</code>
<code>assertTrue(v, msg=None)</code>	<code>bool(v) is True</code>
<code>assertFalse(v, msg=None)</code>	<code>bool(v) is False</code>
<code>assertIs(a, b, msg=None)</code>	<code>a is b</code>
<code>assertIsNot(a, b, msg=None)</code>	<code>a is not b</code>
<code>assertIsNone(v, msg=None)</code>	<code>v is None</code>
<code>assertIsNotNone(v, msg=None)</code>	<code>v is not None</code>
<code>assertIn(a, b, msg=None)</code>	<code>a in b</code>
<code>assertNotIn(a, b, msg=None)</code>	<code>a not in b</code>
<code>assertIsInstance(a, b, msg=None)</code>	<code>isinstance(a, b)</code>
<code>assertNotIsInstance(a, b, msg=None)</code>	<code>not isinstance(a, b)</code>
<code>assertRaises(Exception, function, [function args])</code>	Exception is raised

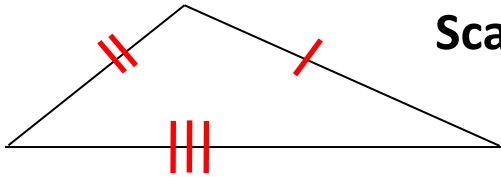
Testing triangle classification



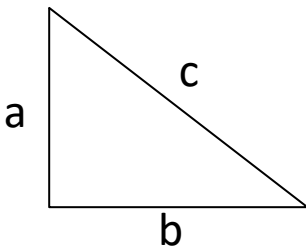
Equilateral: all three sides have the same length



Isosceles: exactly two sides have the same length



Scalene: sides have the three different lengths



Right: $a^2 + b^2 = c^2$



Requirements

“Write a function `classify_triangle()` that takes three parameters: a , b , c representing the lengths of the sides of a triangle. Return a string that specifies whether the triangle is scalene, isosceles, or equilateral, and whether is it a right triangle as well.”

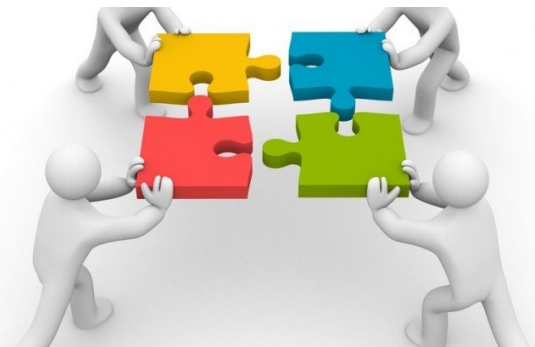
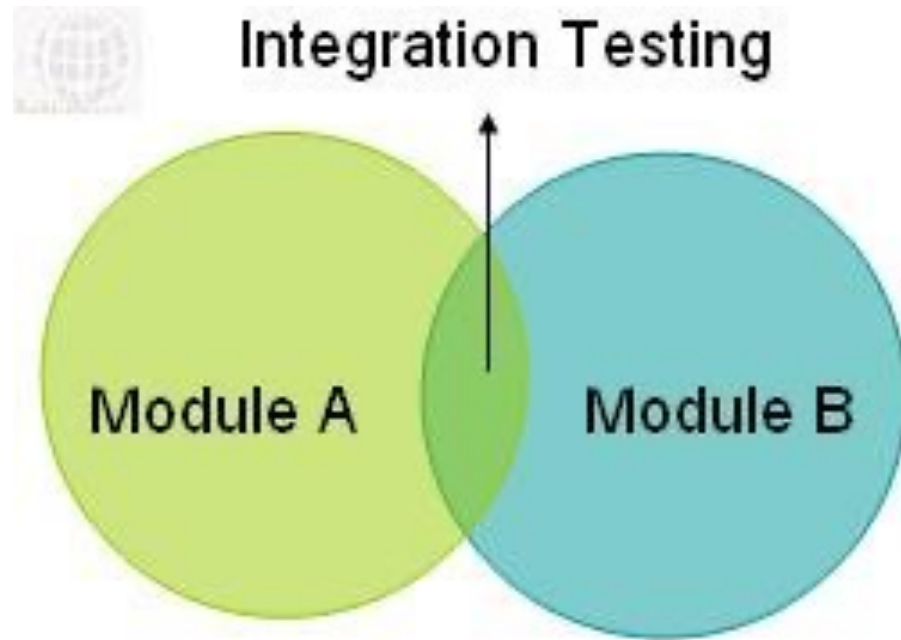
Are these good requirements?

Are they complete?

What's missing from the requirements?

Integration Testing

The purpose of integration testing is to verify the functional, performance, and reliability between the modules that are integrated.



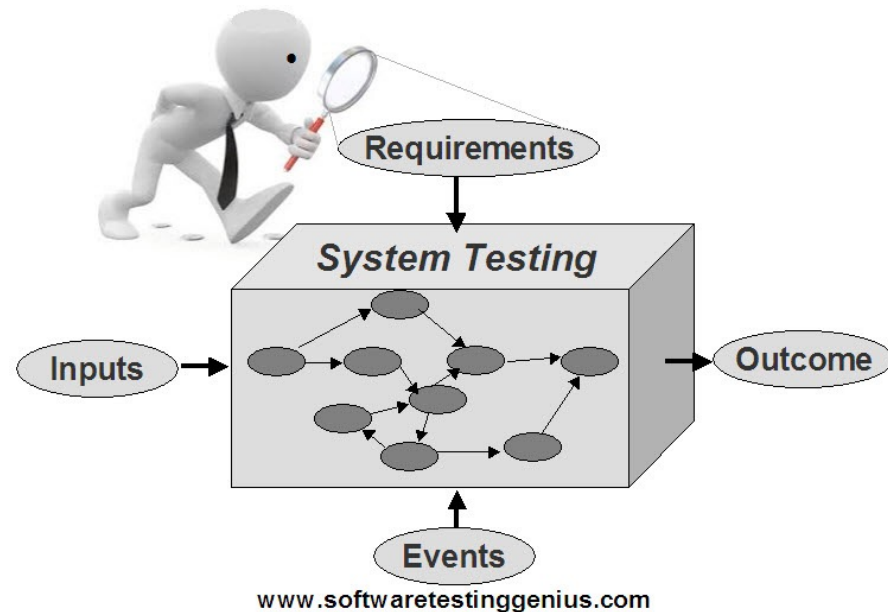
Regression Testing

- Verifies that software, which was previously developed and tested, still performs correctly after it was changed or interfaced with other software.
 - Changes may include software enhancements, patches, configuration changes, bug fixes, etc.
- Techniques:
 - Retest all
 - Test selection
 - Test prioritization



System Testing

- System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements.
- In System testing, the functionalities of the system are tested from an end-to-end perspective.
 - Functionality
 - Performance
 - Scalability
 - Stress
 - Reliability



<http://www.softwaretestinggenius.com/all-about-system-testing-an-important-part-of-our-software-testing-effort>

Acceptance Testing

Formal testing with respect to **user needs**, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.



<http://softwaretestingfundamentals.com/acceptance-testing/>

Software Testing Comparison

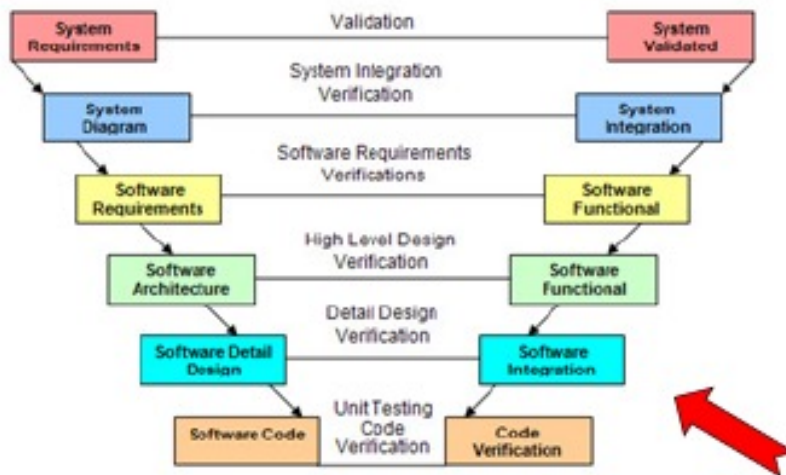
Traditional Methods
React to quality issues



Agile Methods
Proactively reduce
quality issues

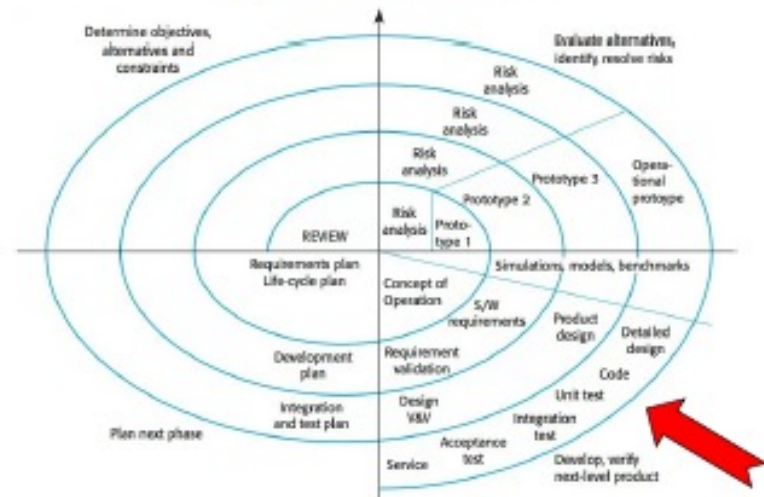
Traditional Software Testing

Waterfall/V Model



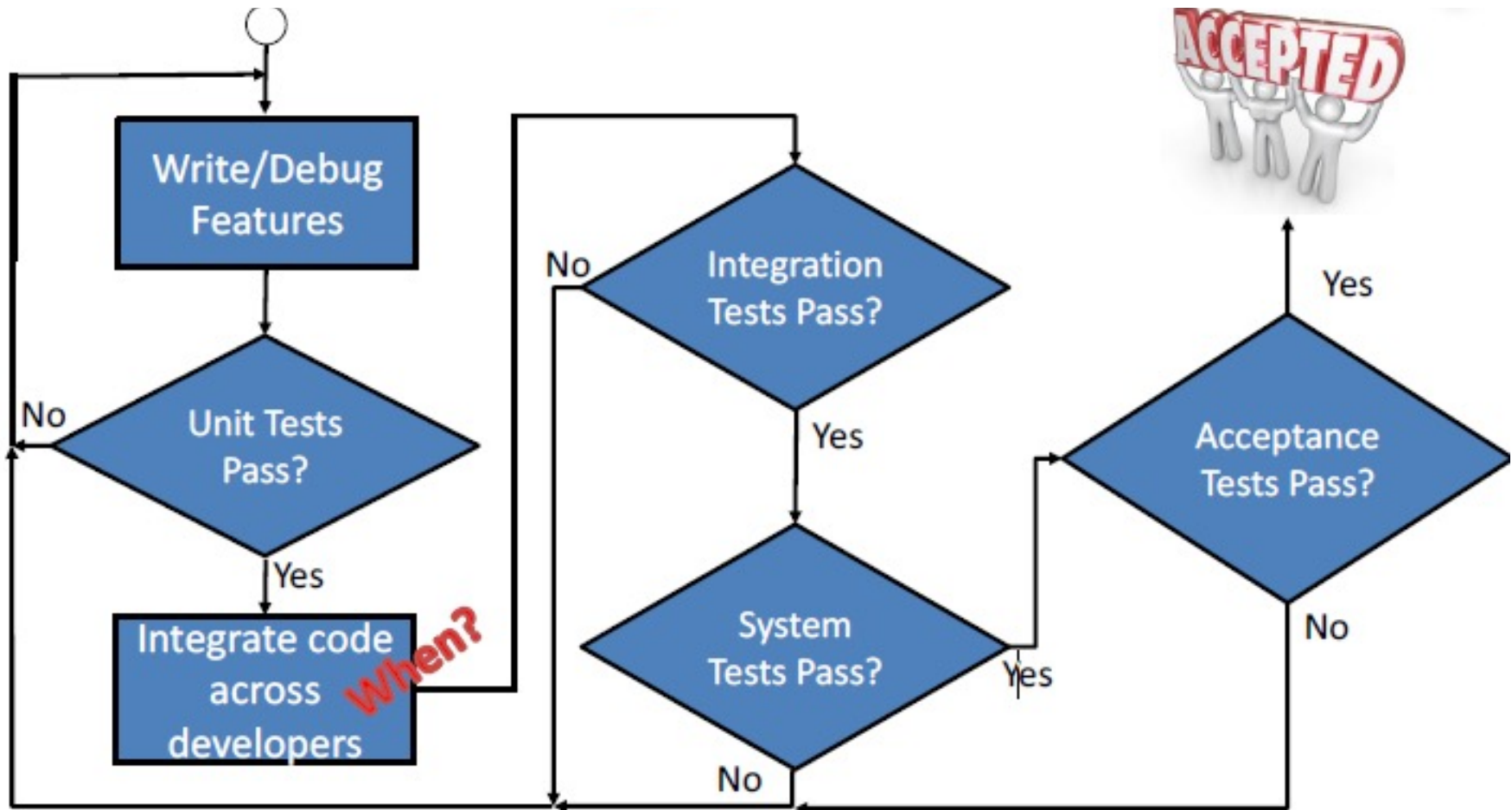
<https://sites.google.com/site/advancedsofteng/software-acquisition/software-development-lifecycle-approaches>

Spiral Model

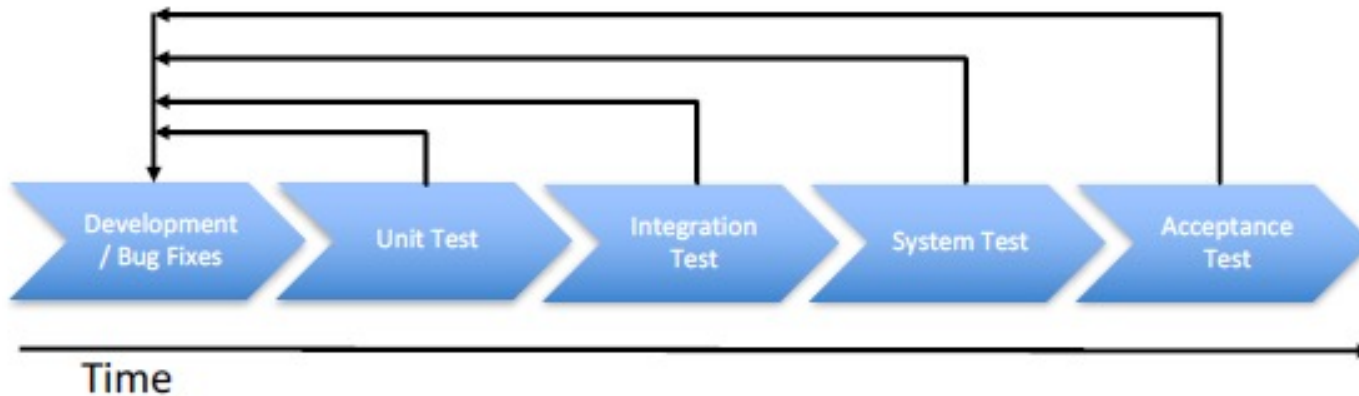


<http://iansommerville.com/software-engineering-book/web/spiral-model/>

Traditional Testing Flow



Traditional Testing



- Testing is blocked until development is “complete”
- Testing is performed by a separate test team
 - Developers might not be trusted to test their own code
- Testing schedule may be compressed if development schedule slips



Agile Testing

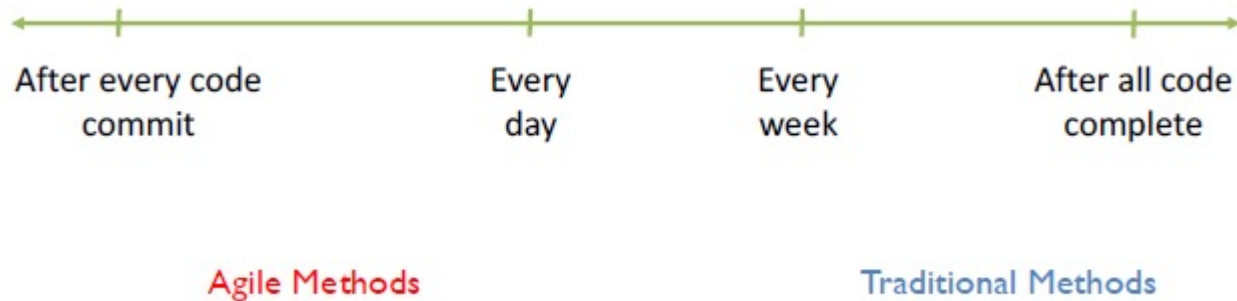
- Testing is NOT a separate phase with Agile Methods
 - Instead, testing is integrated into every step
- Focus on ***automated testing*** and ***continuous integration***
 - Find problems as quickly as possible
 - Expect frequent changes, so facilitate them

Agile testing is not just a passing phase...

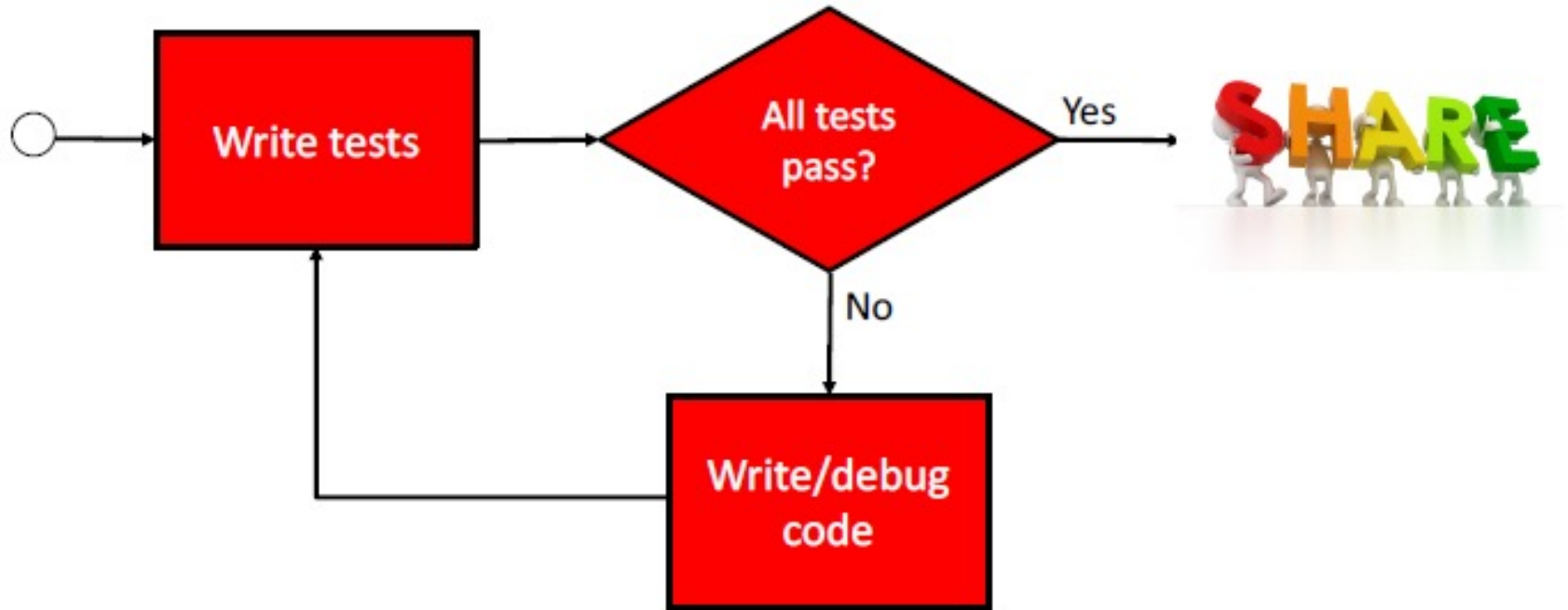
When to integrate?

Integration combines code from multiple developers

How frequently? Continuously? Periodically?



Agile Testing Flow: Automated and Continuous





Unit Testing

	Traditional	Agile
Who?	Developers	Developers using automated Testing platforms
What?	New features or bug fixes; Code/branch coverage	New features or bug fixes code/branch coverage
When?	After code complete; Before integration	Tests written before code is written; Run tests while developing code



Integration Testing

	Traditional	Agile
Who?	Developers and/or testers	Developers using automated testing platforms
What?	Test multiple modules from potentially different developers	Test all modules from all developers
When?	After unit test is complete on relevant modules	Continuously: at least once per day if not on every code check in using existing tests



Regression Testing

	Traditional	Agile
Who?	Testers (Not the developers)	Developers using automated testing platforms
What?	Integrated modules	Entire system or selected components
When?	After changes to code for bug fixes or new features	After changes to code for bug fixes or new features



System Testing

	Traditional	Agile
Who?	System Test Team (not the developers)	Developers using automated testing platforms
What?	Test complete system Performance, security, stress testing	Test complete system Performance, security, stress testing
When?	After all code modules are complete and tested	Optional: may be done before product release cycle after several sprints



Acceptance Testing

	Traditional	Agile
Who?	Customer test engineers	Product Owner, Stakeholders
What?	Complete system based on requirements	Overall system, Including new features from latest sprint
When?	After system test, frequently before final payment	Sprint Review

Customer Bug Reports?

- Agile Methods' focus on automated testing and continuous integration helps to reduce, but doesn't eliminate bugs found by customers
- Scrum adds new features each sprint
 - How are bugs tracked?
 - When are bugs fixed?
- Add bugs as user stories to the product backlog
 - Product owner prioritizes new features and bug fixes



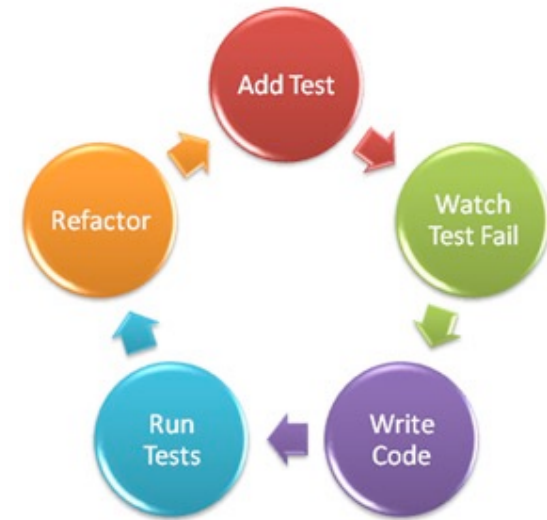


Test-First or Test-Driven Development (TDD)

- Motivation:
 - Programmers don't write tests because:
 - They don't like to
 - They don't have time
 - They have "more important" things to do
 - Result:
 - Code breaks
 - Debugging reduces productivity and doesn't improve testing
 - Still no tests

Alternative TDD Scenario

- Write tests first
- Run the tests (they will probably fail)
- Write some code
- Rerun the tests
- Debug until the tests pass
 - Relatively little untested code at any one time so bugs are likely to be in the most recent code





TDD Provides Useful Feedback

- Programmers get feedback when their tests pass
- Programmers get feedback when their tests fail
- Customers get feedback when the tests pass or fail
 - Provides insights on how the developers are doing
- Provides frequent metrics for management



Pace

- Write a few tests
- Write a few lines of code
- Don't write too many tests at once
- Don't write too much code at once because it's harder to debug until you know it works properly



From TDD to BDD (Behavior-Driven Development)

- Customers and some developers have trouble defining tests:
 - Where to start?
 - What to test?
 - What to call the tests?
- How can we improve the test process to include the customer?
 - Change the syntax and simplify the process!



Simplify Testing

- Replace source code with natural language descriptions
 - Eliminate the word "Test"
- Change from:
 - *public class CustomerLookupTest extends TestCase {*
 - *testFindsCustomerById() { ... }*
 - *testFailsForDuplicateCustomers() { ... }*
 - *...*
 - *}*

- To:

CustomerLookup

finds customer by id

fails for duplicate customers

...



Map User Story to Tests

As a [X]

I want [Y]

so that [Z]

As a customer,

I want to withdraw cash from an ATM,

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



Behavior Templates Aid Testing

Given some initial context (the givens),

When an event occurs,

Then ensure some outcomes.

Given the account is in credit

And the card is valid

And the dispenser contains cash

Pre-conditions

When the customer requests cash

Event

Then ensure the account is debited

And ensure cash is dispensed

And ensure the card is returned

Post-conditions



User Stories to Behavior Stories

- Behavior stories are still easy for customers to write and understand
 - Describe what needs to happen
- BUT...

Behavior Stories are easier to map automatically from user descriptions to executable test cases



BDD Tools

- JBehave: JUnit for customers
 - RBehave: JBehave in Ruby
 - Behave: Behave for Python
 - RSpec: Evolution of RBehave
 - Cucumber: UI specs in natural language for Ruby
- ... similar tools for Python, C, C++, Delphi, PHP, .Net



STEVENS
INSTITUTE *of* TECHNOLOGY
THE INNOVATION UNIVERSITY®

stevens.edu

Thank You