Construx®
Software Development Best Practices

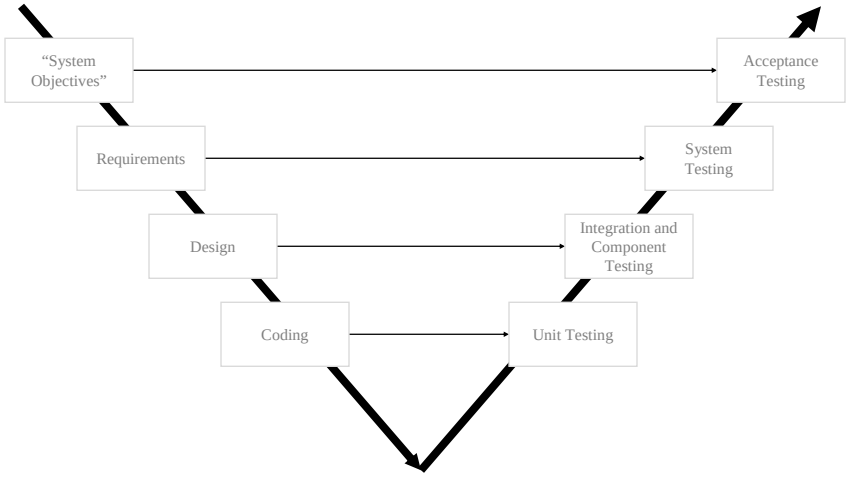
Test Driven Development

http://www.construx.com

Construx®
Software Development Best Practices

Test Driven Development, What is it and Why?

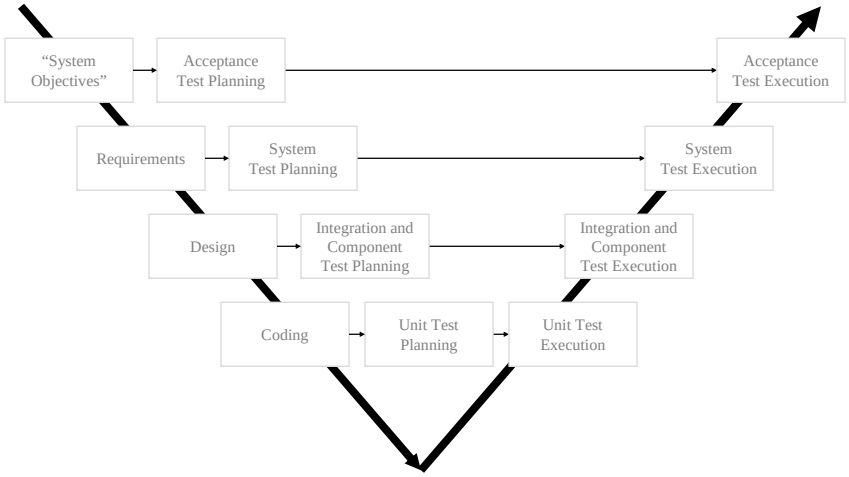
Traditional View of Testing



We Pause to Bring You This Important Message ...

"More than the act of testing, the act of designing tests is one of the best [defect] preventers known … The thought process that must take place to create useful tests can discover and eliminate problems at every stage of development" Boris Beizer

Improved View of Testing

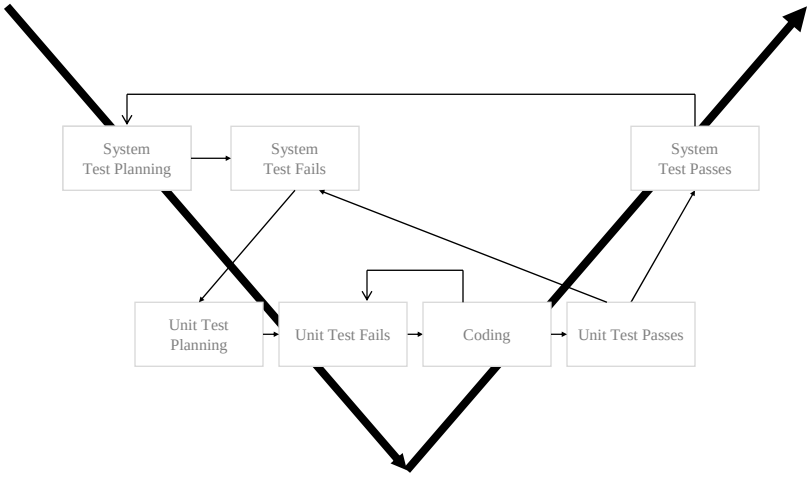


XP's "Test First"

- Create test cases before writing code
  - Business rep writes acceptance tests to demonstrate that user stories are correctly implemented
  - Programmers continually write unit tests which must run flawlessly for development to continue
- "We only write new code when we have a test that doesn't work"

Reference: [Jeffries01]

Test Driven Development's View



Test Driven Development Process

repeat
  select functionality to implement
  create and/or modify system-level tests
  repeat
    developer selects one unit-level module to write and/or modify
    developer creates and/or modifies unit-level tests
    repeat
      developer writes and/or modifies unit-level code
    until all unit-level tests pass
  until all system-level tests pass
until no more functionality to implement

Test A Little, Code A Little

- You don't need to write all the test cases first
  1. You only have to create of one test that the current code won't pass
  2. Write code to pass the test
  3. Go back to step 1
- If it is hard to write a test there may be a design issue
- Each round of test & code includes refactoring of the previous code

Why Test Driven Development?

- One of the biggest problems in software is requirements ambiguity
  - A direct result of using natural language specifications (e.g., "The system shall be fast")
- A test case is inherently unambiguous
  - Test cases are unambiguous "proxies" for requirements

Advantages of Test Driven Development

- Gradually builds an comprehensive suite of (hopefully automated) test cases
  - Run that suite each time the code is compiled
  - All tests must pass except the brand new one(s)
- Code can be refactored with confidence
- Saves time during integration and system testing
  - Most tests can be run automatically
  - Many integration errors can be found before system test

Construx®
Software Development Best Practices

QA-level Test Driven Development

A Use Case Description Template

Use case # nnn	Use case name here
Actor(s)	Identify which actors can access use case
Description	Give an overall description of intent of use case
Preconditions	Identify what must be true at the start of use case to complete successfully
Postconditions	Identify what must be true on completion of use case to complete successfully
Priority	State how important use case is relative to all of other use cases in the system (note: this could be interpreted as either execution priority or development priority)
Normal course	Describe typical execution scenario, if important
Alternative courses	Identify any nontypical execution scenarios that still constitute successful completion of use case. These are different ways that postconditions could still be satisfied
Exceptions	Identify any execution scenarios that constitute unsuccessful completion of this use case. These are different ways that, despite the preconditions having been satisfied, the postconditions will not have been achieved.
Special requirements	List any other specific (i.e., nonfunctional) requirements that apply to use case
Assumptions	Identify any assumptions behind specification of use case

Example Use Case

Use case # 66	Make Flight Reservation(s)
Actor(s)	Travel Agent, Traveler, Airline Ticket Agent
Description	Make a reservation on one or more requested flight segments in name of specified traveler.
Preconditions	Each specified flight segment exists. There is room available in fare/class for each flight. All applicable advance-purchase/stay requirements are satisfied.
Postconditions	Each reservation instance has been created. The reservation confirmation has been given to actor. Space available in each fare/class for each flight has been reduced.
Priority	Highest
Normal course	Reservation request is made and completed.
Alternative courses	Actor pays for reservations immediately. Actor preassigns seats. Actor requests special meal. Actor requests extra service like wheelchair or unaccompanied minor service (traveler is under 12 years old).
Exceptions	One or more minimum airport connection times isn't satisfied. Traveler is on the FBI watch list.
Special requirements	Must be completed in under 60 seconds.
Assumptions	This use case only covers making reservations for one person at a time. It also doesn't address related travel services like rental car and hotel reservations.

Testing from Use Cases

- Positive tests
  - Normal course
  - Each alternative course
  - Combinations of alternative courses?
  - Validate the assumptions?
- Negative tests
  - Violate each precondition
  - Force each exception

Example Use Case-based Tests

- TC1 (Positive, normal course)
  - Reservation on existing flight(s) with room in fare/class and advance purchase/stay requirements met → reservation created, confirmation provided, availability reduced
- TC2 (Positive, alternative course 1)
  - Reservation on existing flight(s) with room in fare/class and advance purchase/stay requirements met & pay immediately → …
- TC3 (Positive, alternative course 2)
  - Reservation on existing flight(s) with room in fare/class and advance purchase/stay requirements met & pre-assign seat(s) → …
- TC4 (Positive, alternative course 3)
  - Reservation on existing flight(s) with room in fare/class and advance purchase/stay requirements met & special meal→ …
- TC5 (Positive, alternative course 4)
  - Reservation on existing flight(s) with room in fare/class and advance purchase/stay requirements met & unaccompanied minor→ …
- TC6 → 16 (Positive, all other combinations of alternative courses)
  - …
  - Reservation on existing flight(s) with room in fare/class and advance purchase/stay requirements met & pay now & pre-assign seat(s) & special meal & unaccompanied minor→ reservation created, confirmed, availability--, paid, seat assigned, meal rqst'd, and UM

Example Use Case-based Tests (cont)

- TC17 (Negative, violate precondition 1)
  - Reservation on non-existing flight(s) → "Flight doesn't exist"
- TC18 (Negative, violate precondition 2)
  - Reservation on existing flight(s) and advance purchase/stay requirements met but no room in fare/class → "No room in fare/class"
- TC19 (Negative, violate precondition 3)
  - Reservation on existing flight(s) with room in fare/class but advance purchase/stay requirements not met → "Advance purchase/stay not met"
- TC20 (Negative, force exception 1)
  - Reservation on existing flights with room in fare/class and advance purchase/stay requirements met but minimum connect time not met → "Minimum connect time not met"
- TC21 (Negative, force exception 2)
  - Reservation on existing flight(s) with room in fare/class and advance purchase/stay requirements met but traveler on FBI watch list → "Traveler on FBI Watch List"

Construx®

Software Development Best Practices

Developer-level Test Driven Development

Example of Test First Development

- Suppose we need a method that finds the largest number in an array

```
int Largest.largest(int[] list);
```

- Given [7, 8, 9] it should return 9

From Andrew Hunt and David Thomas, *Pragmatic Unit Testing: In Java with Junit, 2003*

Example Test Cases

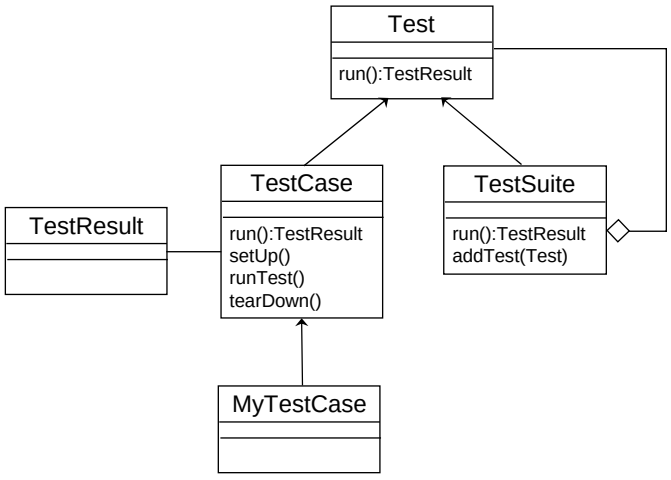
- [7, 8, 9] → 9
- [8, 9, 7] → 9 ***Order shouldn't matter***
- [9, 7, 8] → 9
- [7, 9, 8, 9] → 9 ***Multiple occurrences are OK***
- [1] → 1 ***One element is still an array***
- [-9, -8, -7] → -7 ***Negative numbers are OK***

Unit Test Frameworks

- Automates unit-level testing
  - Write unit tests in the same language you are coding in
- Many open source, downloadable, free frameworks available
  - JUnit for Java
  - NUnit for C#
  - CppUnit for C++
  - …

Links at www.xprogramming.com/software.htm

JUnit Design



Asserts in JUnit

- assertEquals()
- assertFalse()
- assertTrue()
- assertSame()
- assertNotSame()
- assertNull()

JUnit Test Code (1)

```
import junit.framework.*
public class TestLargest extends TestCase {
  public TestLargest(String name) {
    super(name);
  }
  public void testSimple() {
    assertEquals(9,
      Largest.largest(new int[ ] {7, 8, 9}));
  }
}
```

What happens when we run this test?

Initial Code for the Method

```
public class Largest {
  public static int largest(int[] list) {
    int index, max = Integer.MAX_VALUE;
    for (index = 0; index < list.length-1; index++){
      if (list[index] > max) {
        max = list[index];
      }
    }
  return max;
  }
}
```


What happens when we run the test?

```
There was 1 failure:
1)testSimple(TestLargest)junit.framework.AssertionF
  ailedError: expected: <9> but was: <2147483647>
  at TestLargest.testSimple(TestLargest.java:11)
```

- Assignment

```
max = Integer.MAX_VALUE
```

- should have been more like

```
max=0
```

Change the code, run the test, now it passes

JUnit Test Code (2)

```
import junit.framework.*
public class TestLargest extends TestCase {
  public TestLargest(String name) {
    super(name);
  }
  public void testSimple() {
    assertEquals(9,
      Largest.largest(new int[ ] {7, 8, 9}));
  }
  public void testOrder() {
    assertEquals(9,
      Largest.largest(new int[] {9, 8, 7}));
    assertEquals(9,
      Largest.largest(new int[] {7, 9, 8}));
}
```


What happens when we run the tests?

```
There was 1 failure:
1)testOrder(TestLargest)junit.framework.AssertionF
  ailedError: expected: <9> but was: <8> at
  TestLargest.testOrder(TestLargest.java:10)
```

- Ignoring last item in the list

```
for (index = 0; index < list.length-1; index++)
```

- should be

```
for (index = 0; index < list.length; index++)
```

Change the code, run the tests, now they pass

JUnit Test Code (3)

```
import junit.framework.*
public class TestLargest extends TestCase {
  public TestLargest(String name) {
     super(name);
  }

  // leaving out tests already shown

  public void testDups() {
    assertEquals(9,
      Largest.largest(new int[ ] {9, 7, 9, 8}));
  }
  public void testOne() {
    assertEquals(1, Largest.largest(new int[] {1}));
  }
  public void testNegative() {
    int [] negList = new int[] {-9, -8, -7};
    assertEquals(-7, Largest.largest(negList));
  }
}
```


What happens when we run the tests?

```
There was 1 failure:
1)testNegative(TestLargest)junit.framework.Assertio
  nFailedError: expected: <-7> but was: <0> at
  TestLargest.testNegative(TestLargest.java:16)
```

- 0 is bigger than any negative number, so it will be returned as the largest, want

```
max = Integer.MIN_VALUE
```

Change the code, run the tests, now they pass

Transitioning to Test Driven Development

Transitioning to Test Driven Development

- Don't try to write tests for the whole thing!
  - Write tests for the parts you are adding or changing
  - Write tests for parts that are causing you problems
  - Gradually you'll build up a set of tests
- You may find the code isn't designed to make writing tests easy
  - May have to be refactored or rewritten

Contact Information

Consulting@construx.com
www.construx.com
(425) 636-0100

Construx

Software Development Best Practices

- Consulting
- Seminars

sales@construx.com
www.construx.com