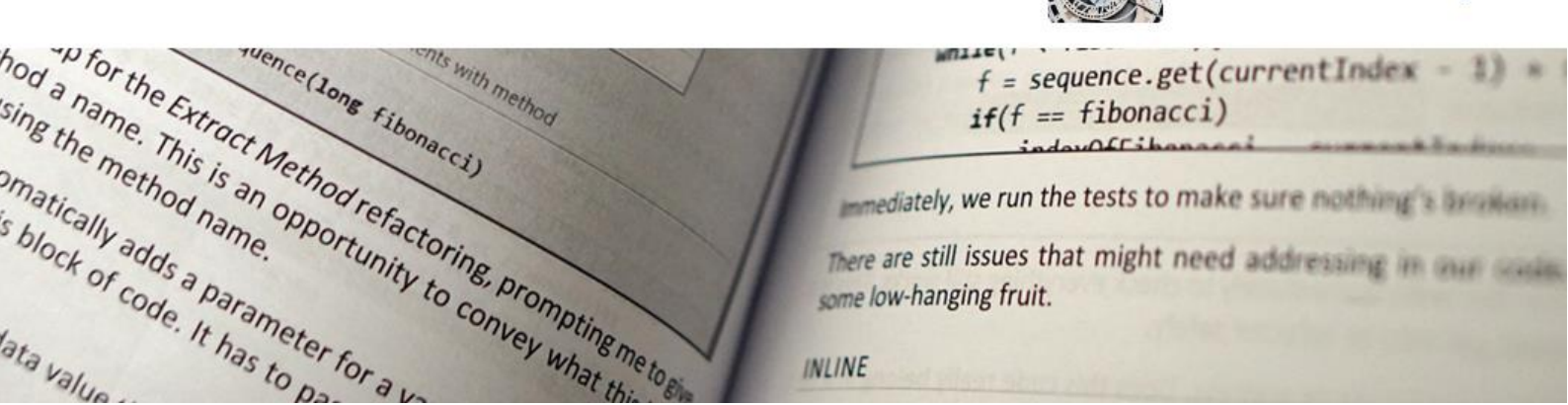


codemanship



# 101 TDD Tips

101 things you'll learn on the Codemanship TDD 2.0 training workshop.  
First published on the @codemanship Twitter feed.

Visit [www.codemanship.com](http://www.codemanship.com) for more details about TDD training and coaching, and our exclusive 200-page TDD book.

### About The Author



Jason Gorman is a software developer, trainer and coach based in London. A TDD practitioner since before it had a name, he's helped thousands of developers to learn this essential discipline through his company *Codemanship*. He's the founder of the original international *Software Craftsmanship 20xxx* conference, an activist for software developer apprenticeships, a patron of the Bletchley Park Trust, a one-time-only West End producer, a failed physicist, and a keen amateur musician. His twelve fans know him as *Apes With Hobbies*.

You can follow him on Twitter (@jasongorman), or email [jason.gorman@codemanship.com](mailto:jason.gorman@codemanship.com)

### About Codemanship



**codemanship**

Founded in 2009, Codemanship provides training, coaching and consulting in the practical software disciplines that enable organisations to sustain the pace

of digital innovation. Based in London, Codemanship has trained and guided teams in TDD, refactoring, software design, Continuous Integration and Continuous Delivery, and Agile Software Development for a wide range of clients including the BBC, UBS, Waters plc, Ordnance Survey, salesforce.com, Electronic Arts, John Lewis, Redgate and Sky.

**TDD Tip #1:** Refactoring to parameterised tests is a great way to reduce duplication while generalising the tests so they read more like a specification

```
@RunWith(JUnitParamsRunner.class)
public class FibonacciTests {

    @Test
    @Parameters({"0,0", "1,1"})
    public void startsWithZeroAndOne(int index, int expected) {
        assertEquals(expected, getFibonacciNumber(index));
    }

    @Test
    @Parameters({"2,1", "3,2", "5,5"})
    public void thirdNumberOnIsSumOfPreviousTwo(int index, int expected){
        assertEquals(expected, getFibonacciNumber(index));
    }

    @Test(expected=IllegalArgumentException.class)
    public void indexMustBePositiveInteger() {
        getFibonacciNumber(-1);
    }

    private int getFibonacciNumber(int index) {
        return new Fibonacci().getNumber(index);
    }
}
```



codemanship



**TDD Tip #2:** Using variables and constants can make the meaning of test data values clearer

```
@Test(expected=MaximumExceededException.class)
public void maximumDebitAmountCannotBeExceeded() {
    BankAccount account = new BankAccount();
    account.credit(1000);
    account.debit(600.01);
}
```

```
@Test(expected=MaximumExceededException.class)
public void maximumDebitAmountCannotBeExceeded() {
    BankAccount account = new BankAccount();
    account.credit(1000);
    final double maxDebitAmount = 600.00;
    account.debit(maxDebitAmount + 0.01);
}
```



codemanship

**TDD Tip #3:** It's actually okay to have getters for tests, just as long as they're not exposed to the client source code

```
@Test
public void rewardingMemberAddsPointsToTotal() {
    Member member = new Member();
    member.reward(10);
    assertEquals(10, member.getRewardPoints());
}

public class Member implements Rewardable {

    private int rewardPoints;

    @Override
    public void reward(int points) {
        this.rewardPoints += points;
    }

    public int getRewardPoints() {
        return rewardPoints;
    }
}

public class Library {

    private List<Copyable> titles;

    public void donate(Copyable title, Rewardable donor) {
        titles.add(title);
        donor.reward(10);
    }
}
```



codemanship

**TDD Tip #4:** Running customer tests through a tag cloud generator can provide inspiration when looking for names for classes, methods, variables etc

**Given** a movie title that isn't in the library,

**When** a member donates their copy

**Then** the title is added to the library,

And a default loan copy is added to the title,

And an email alert is sent to all members who expressed an interest in matching titles informing them title is available to borrow,

And the donor is awarded 10 reward points



codemanship

**TDD Tip #5:** You don't necessarily need a mocking framework to create mock objects

```
public class LibraryTests {  
  
    private boolean registerCopyInvoked;  
  
    @Test  
    public void tellsTitleToRegisterCopy() {  
        registerCopyInvoked = false;  
        Member member = new Member() { public void awardPriorityPoints(int points) {}  
        };  
        Title title = new Title() {  
            public void registerCopy() {  
                registerCopyInvoked = true;  
            }  
        };  
        new Library().donate(title, member);  
        assertTrue("title.registerCopy() was not invoked", registerCopyInvoked);  
    }  
}
```



codemanship

**TDD Tip #6:** The way to go faster is to go cleaner. When the schedule's slipping, consider taking *smaller* steps

```
@Test
public void squareRootTest() {
    assertEquals(3, Maths.sqrt(9), 0.00001);
}
```

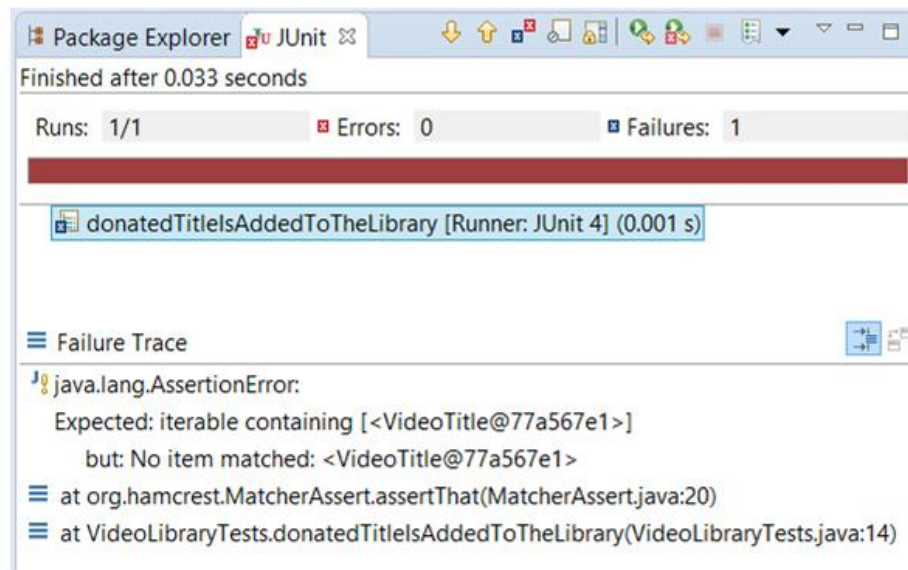


```
@Test
@Parameters({"0", "1", "4", "9", "0.25"})
public void squareOfSquareRootIsSameAsInput(double input) {
    double sqrt = Maths.sqrt(input);
    assertEquals(input, sqrt * sqrt, 0.00001);
}
```



codemanship

**TDD Tip #7:** Before you make it pass, run the test to make sure the assertion fails when the result is wrong, so you know it's a good test



codemanship



## TDD Tip #8: Customer Tests Passed offers a more objective measure of progress than 'tasks completed' or 'layers coded'

Feature	Progress %	UI	Services	Domain	DB
Donate a DVD	70%	0%	80%	100%	100%
Borrow a DVD	75%	0%	100%	100%	100%
Join the library	65%	0%	60%	100%	100%
Refer a friend	75%	0%	100%	100%	100%
Review a movie	75%	0%	100%	100%	100%
Search for titles	50%	0%	0%	100%	100%
Report DVD lost or damaged	50%	0%	0%	100%	100%
Reverse a DVD	50%	0%	10%	100%	100%
Spend reward points	75%	0%	10%	100%	100%
Transfer reward points	75%	0%	10%	100%	100%
<b>Total progress</b>	<b>66%</b>				

Feature	Progress %	Total Tests	Passed
Donate a DVD	60%	5	3
Borrow a DVD	100%	4	4
Join the library	100%	2	2
Refer a friend	100%	2	2
Review a movie	100%	4	4
Search for titles	0%	4	0
Report DVD lost or damaged	0%	2	0
Reserve a DVD	0%	2	0
Spend reward points	100%	2	2
Transfer reward points	100%	1	1
<b>Total progress</b>	<b>66%</b>		

Feature	Progress %	Analysis	Design	UI	Services	Domain	DB
Donate a DVD	75%	100%	100%	100%	80%	100%	100%
Borrow a DVD	75%	100%	100%	100%	100%	100%	100%
Join the library	68%	100%	100%	100%	60%	100%	100%
Refer a friend	70%	100%	100%	100%	100%	100%	100%
Review a movie	50%	100%	100%	100%	100%	100%	100%
Search for titles	50%	100%	100%	100%	0%	100%	100%
Report DVD lost or damaged	63%	100%	100%	100%	50%	100%	100%
Reverse a DVD	63%	100%	100%	100%	50%	100%	100%
Spend reward points	75%	100%	100%	100%	100%	100%	100%
Transfer reward points	75%	100%	100%	100%	100%	100%	100%
<b>Total progress</b>	<b>66%</b>						



codemanship

**TDD Tip #9:** Parameterised unit tests can be reused in other kinds of test fixtures that can be run separately

```
@RunWith(JUnitParamsRunner.class)
public class MathsTests {

    @Test
    @Parameters({"0", "1", "4", "9", "0.25"})
    public void squareOfSquareRootIsSameAsInput(double input) {
        double sqrt = Maths.sqrt(input);
        assertEquals(input, sqrt * sqrt, 0.00001);
    }
}

@RunWith(JUnitParamsRunner.class)
public class ExhaustiveMathsTests {

    @Test
    @Parameters(method="inputs")
    public void test1000SquareRoots(double input) {
        new MathsTests().squareOfSquareRootIsSameAsInput(input);
    }

    private Object[] inputs(){
        return DoubleStream
            .iterate(1, n -> n + 0.1)
            .limit(1000)
            .mapToObj(x -> x)
            .toArray();
    }
}
```



codemanship

**TDD Tip #10:** When the implementation for a requirement or rule is obvious, you don't need to triangulate through multiple examples

```
@Test
public void sumOfTwoNumbers() {
    assertEquals(4, Maths.sum(2,2), 0);
}

public class Maths {

    public static double sum(double i, double j) {
        return i + j;
    }
}
```



codemanship

**TDD Tip #11:** Don't mock or stub 3<sup>rd</sup> party interfaces. Create your own interfaces that you control to simplify interactions and protect your code from changes

```
@Test
public void whenOrderConfirmedNotifiesWarehouse() throws IOException {
    com.rabbitmq.client.Channel warehouseChannel
        = mock(com.rabbitmq.client.Channel.class);

    Order order = new Order(warehouseChannel);
    Product product = new Product("Widget", 9.99);
    order.addItem(new OrderItem(product, 1));

    order.confirm();

    String message = "NEW ORDER\nItem 1: Widget, Quantity: 1";

    verify(warehouseChannel)
        .basicPublish("", "WAREHOUSE", null, message.getBytes());
}

@Test
public void whenOrderConfirmedNotifiesWarehouse() {
    Warehouse warehouse = mock(Warehouse.class);

    Order order = new Order(warehouse);
    Product product = new Product("Widget", 9.99);
    order.addItem(new OrderItem(product, 1));

    order.confirm();

    verify(warehouse)
        .notify(order);
}
```



codemanship

**TDD Tip #12:** Tests should have one reason to fail, so we can more easily pinpoint failures, and get feedback one design decision at a time

```
@Test
public void donateTitle() {
    Library library = new Library();
    VideoTitle title = new VideoTitle();
    Member donor = mock(Member.class);
    library.donate(title, donor);
    assertTrue(library.contains(title));
    verify(donor).awardPoints(10);
}
```



```
private Library library;
private VideoTitle title;
private Member donor;
```

```
@Before
public void donateTitle() {
    library = new Library();
    title = new VideoTitle();
    donor = mock(Member.class);
    library.donate(title, donor);
}
```

```
@Test
public void donatedTitleIsAddedToLibrary() {
    assertTrue(library.contains(title));
}
```

```
@Test
public void tellsDonorToAwardTenPoints() {
    verify(donor).awardPoints(10);
}
```



codemanship

**TDD Tip #13:** Writing the test assertion first and working backwards to the setup helps us to focus on the “what” before the “how”

```
@Test  
public void donatedTitleIsAddedToLibrary() {  
    assertTrue(library.contains(title));  
}
```

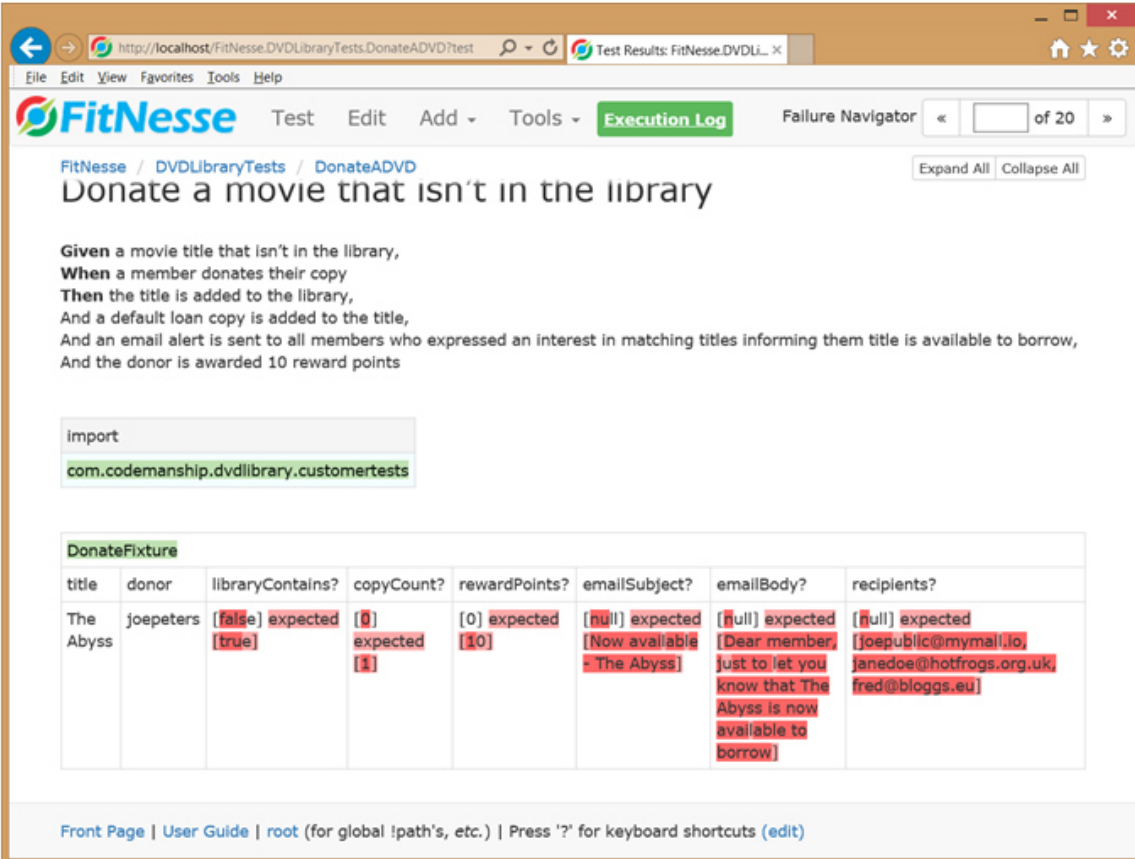
- 🔍 Create local variable 'library'
- ▣ Create field 'library'
- 🔍 Create parameter 'library'
- 🟢 Create class 'library'
- ▣ Create constant 'library'
- 🔍 Rename in file (Ctrl+2, R)
- 🔍 Convert to TestNG (Annotations)
- 🔍 Pull @Test annotations to the class level
- ➡ Change to 'Library'
- ➡ Change to 'LibraryTests'
- ➡ Fix project setup...



codemanship



## TDD Tip #14: Start on the outside with a failing customer test, and then test-drive the internal design to pass that customer test



The screenshot shows the FitNesse web interface in a browser. The URL is `http://localhost/FitNesse.DVDLibraryTests.DonateADVD?test`. The page title is "Donate a movie that isn't in the library". Below the title, there is a Gherkin-style test description:

**Given** a movie title that isn't in the library,  
**When** a member donates their copy  
**Then** the title is added to the library,  
 And a default loan copy is added to the title,  
 And an email alert is sent to all members who expressed an interest in matching titles informing them title is available to borrow,  
 And the donor is awarded 10 reward points

Below the description, there is an "import" section with the path `com.codemanship.dvdlibrary.customertests`.

The "DonateFixture" table shows the test results:

title	donor	libraryContains?	copyCount?	rewardPoints?	emailSubject?	emailBody?	recipients?
The Abyss	joepeters	[false] expected [true]	[0] expected [1]	[0] expected [10]	[null] expected [Now available - The Abyss]	[null] expected [Dear member, just to let you know that The Abyss is now available to borrow]	[null] expected [joepublic@mymail.io, janedoe@hotfrogs.org.uk, fred@bloggs.eu]

At the bottom of the page, there is a footer with links: [Front Page](#) | [User Guide](#) | [root](#) (for global !path's, etc.) | Press '?' for keyboard shortcuts ([edit](#))



codemanship

**TDD Tip #15:** There's often a simpler failing test to start with than you think

```
@Test
public void basketTotalledCorrectly() {
    ShoppingBasket basket = new ShoppingBasket();
    Product widget = new Product("Widget", 60.0);
    basket.add(new Item(widget, 2));
    Product flange = new Product("Flange", 30.0);
    basket.add(new Item(flange, 1));
    assertEquals(150.0, basket.getTotal(), 0);
}

@Test
public void emptyBasketHasTotalOfZero() {
    assertEquals(0.0, new ShoppingBasket().getTotal(), 0);
}
```



codemanship

**TDD Tip #16:** Instead of multiple stub implementations, parameterise stubs for less code and so test data can be defined inside the test method

```
public class VideoTitleTests {

    private final float PRICE = 3.95f;

    @Test
    public void titlesRatedEightOrHigherAddDollarPremium() {
        VideoTitle title =
            createTitle(new ImdbStubRatingEight(), PRICE);
        assertEquals(PRICE + 1.0, title.totalPrice(), 0);
    }

    @Test
    public void titlesRatedLessThanEightAddNoPremium() {
        VideoTitle title =
            createTitle(new ImdbStubRatingSeven(), PRICE);
        assertEquals(PRICE, title.totalPrice(), 0);
    }

    private VideoTitle createTitle(ImdbService imdb, float price) {
        return new VideoTitle(imdb, price);
    }
}
```



```
public class ImdbStub implements ImdbService {

    private int rating;

    public ImdbStub(int rating) {
        this.rating = rating;
    }

    @Override
    public float rating() {
        return rating;
    }
}

@Test
public void titlesRatedEightOrHigherAddDollarPremium() {
    VideoTitle title =
        createTitle(new ImdbStub(8), PRICE);
    assertEquals(PRICE + 1.0, title.totalPrice(), 0);
}

@Test
public void titlesRatedLessThanEightAddNoPremium() {
    VideoTitle title =
        createTitle(new ImdbStub(7), PRICE);
    assertEquals(PRICE, title.totalPrice(), 0);
}
```



codemanship

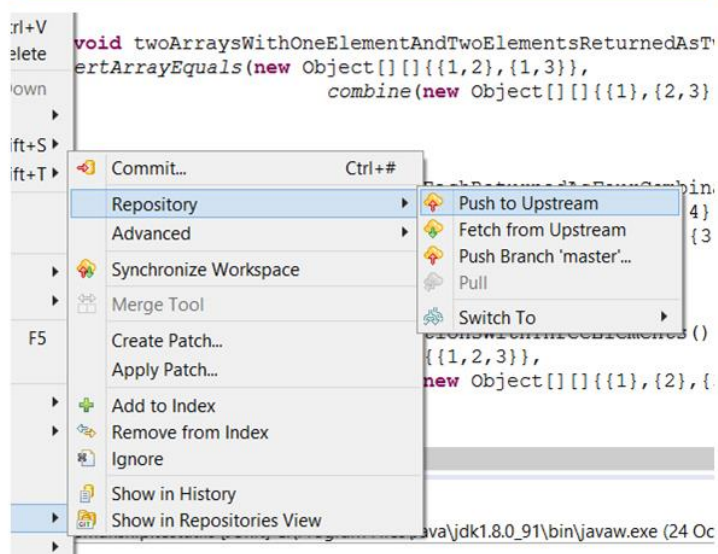
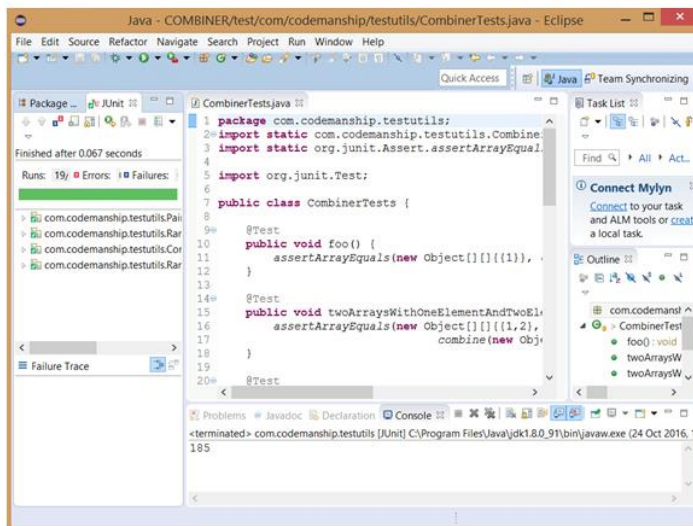
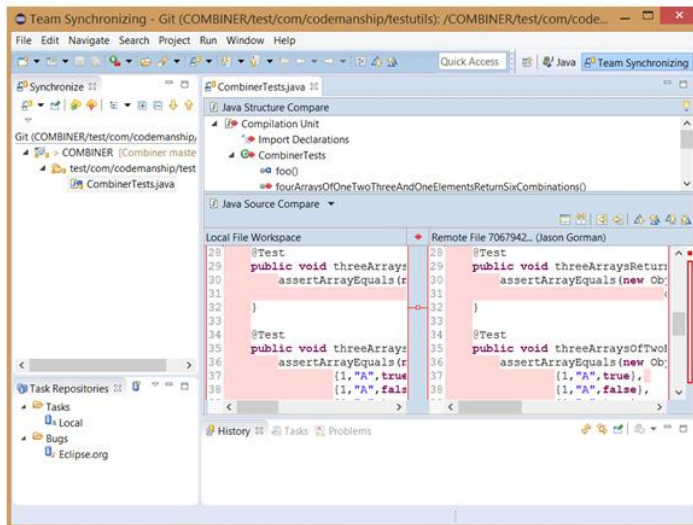
**TDD Tip #17:** The best source of test examples is the real world. Observe people in situations where your software will be used.



codemanship



**TDD Tip #18:** Before you commit, merge other people's changes into your code and make sure all the tests pass



codemanship

**TDD Tip #19:** Avoid using mocking “power” tools to get around dependency issues. You’ll bake in a bad design. Instead, refactor to make the dependency swappable.

```
@Test
public void addDollarForTitlesRatedNineOrMore() {
    String titleId = "tt0111161";
    // stub static method
    PowerMockito
        .stub(PowerMockito.method(ImdbService.class, "fetchRating"))
        .toReturn(9.2);
    VideoRental rental = new VideoRental(titleId);
    assertEquals(4.95, rental.getPrice(), 0.0);
}
```



```
@Test
public void addDollarForTitlesRatedNineOrMore() {
    String titleId = "tt0111161";
    ImdbService imdb = Mockito.mock(ImdbService.class);
    Mockito.when(imdb.fetchRating(titleId)).thenReturn(9.2);
    VideoRental rental = new VideoRental(imdb, titleId);
    assertEquals(4.95, rental.getPrice(), 0.0);
}
```



codemanship



**TDD Tip #20:** The failing tests you start with don't have to be functional

```
public class QuicksortTimeComplexityTests {  
  
    @Test  
    public void isNLogNComplex() {  
        for(int N = 2; N < 1000000; N++){ // time for a coffee!  
            int[] array = buildRandomArray(N);  
            Quicksort quicksort = new Quicksort();  
            quicksort.sortAsc(array);  
            assertTrue(quicksort.getIterations() <= N * Math.log(N));  
        }  
    }  
  
    private int[] buildRandomArray(int length) {  
        int[] array = new int[length];  
        for (int i = 0; i < array.length; i++) {  
            array[i] = (int) (Math.random() * length);  
        }  
        return array;  
    }  
}
```



codemanship

**TDD Tip #21:** Slow-running customer test fixtures can often be adapted to double as fast-running developer tests

```
public class DonateFixture {
    private Library library;
    private Title title;
    private Member donor;
    private EmailQueue queue;
    private ArgumentCaptor<EmailAlert> alert;
    private InterestedMemberSearch search;

    public DonateFixture() {}

    @Test
    public void donateMovieThatIsntInTheLibrary() {
        setTitle("The Abyss");
        setDonor("joepeters");
        assertTrue(libraryContains());
        assertEquals(1, copyCount());
        assertEquals(10, rewardPoints());
        assertEquals("Now available - The Abyss", emailSubject());
        assertEquals("Dear member, just to let you know that", emailBody());
        assertEquals("joepublic@mymail.io, janedoe@hotmail.com", recipients());
    }

    public void setDonor(String memberId) {}

    public void setTitle(String name) {}
}
```

FitNesse fixture



codemanship

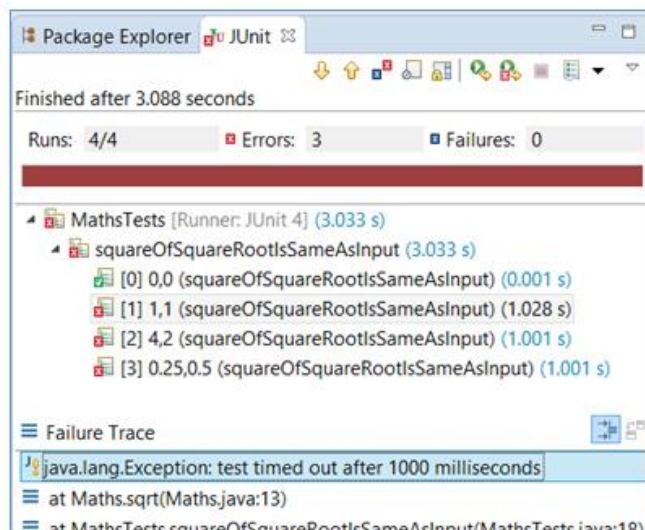
**TDD Tip #22:** Not sure how much faith you can put in your tests to catch new bugs? Mutation testing can help.

```
do {
    t = squareRoot;
    squareRoot = (t + (number / t)) / 2;
} while ((t - squareRoot) != 0);
```

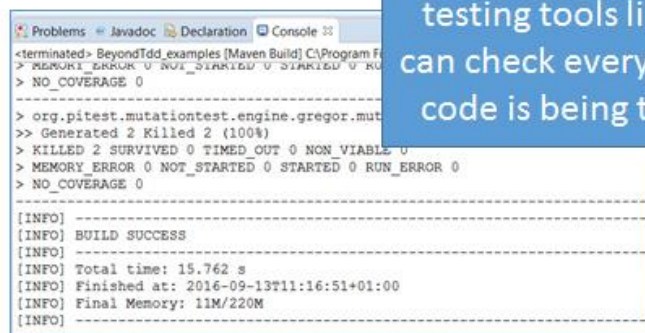
'mutate' + into -

```
do {
    t = squareRoot;
    squareRoot = (t + (number / t)) / 2;
} while ((t - squareRoot) != 0);
```

## Run the tests



Automated mutation testing tools like PIT can check every line of code is being tested



codemanship

**TDD Tip #23:** As you triangulate a solution, your tests should be getting more general, too.

```
@Test
public void rootOfZeroIsZero() {
    assertEquals(0, Maths.sqrt(0), 0.00001);
}
```



```
@Test
@Parameters({"0,0", "1,1"})
public void squareRootTest(double input, double expected) {
    assertEquals(expected, Maths.sqrt(input), 0.00001);
}
```



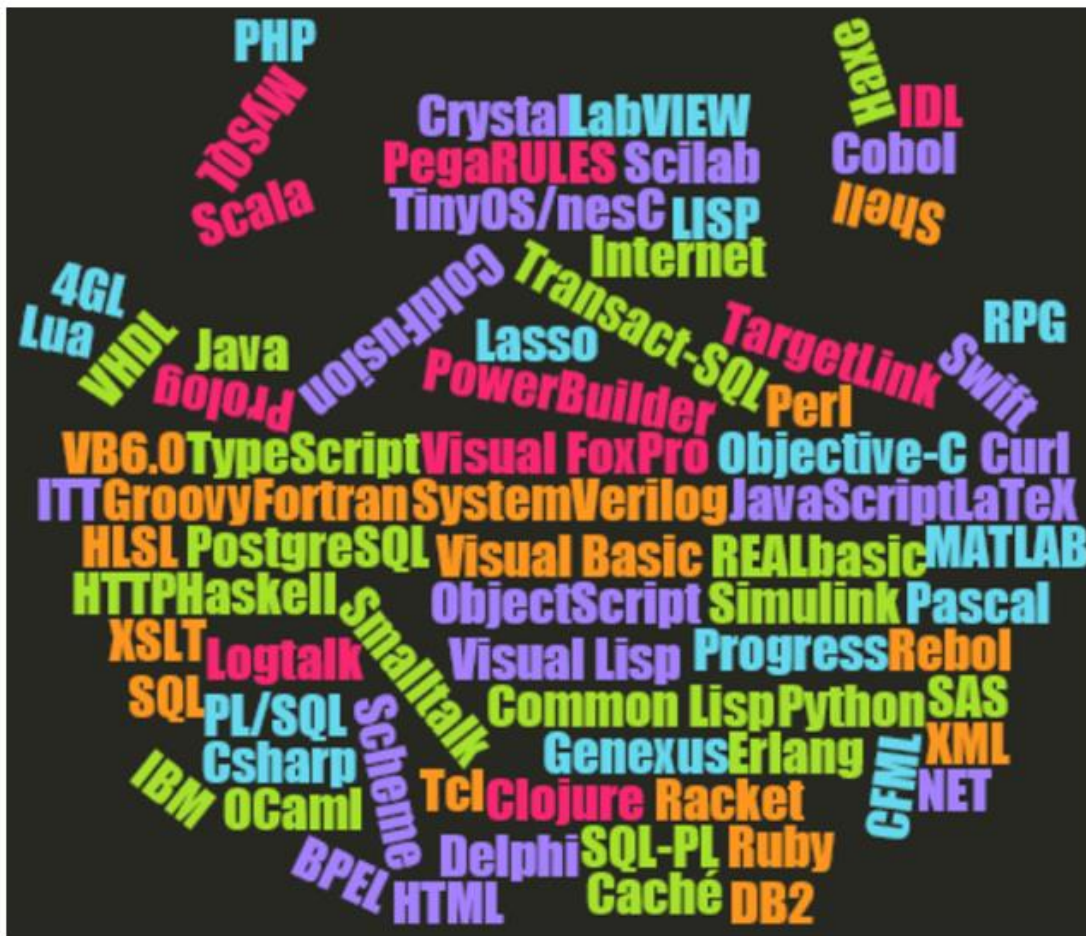
```
@Test
@Parameters({"0", "1", "4", "9", "16", "0.25"})
public void squareOfRootSameAsInput(double input) {
    double root = Maths.sqrt(input);
    assertEquals(input, root * root, 0.00001);
}
```



codemanship



**TDD Tip #24:** Testing tools for TDD exist in most languages. And it only takes a day or two to write a basic framework in most.



codemanship

**TDD Tip #25:** Writing the test assertion first and working backwards can also apply to interaction tests.

```
@Test  
public void tellsDonatedTitleToAddLoanCopy() {  
    verify(title).addLoanCopy();  
}
```

title cannot be resolved to a variable

4 quick fixes available:

- [Create local variable 'title'](#)
- [Create field 'title'](#)
- [Create parameter 'title'](#)
- [Create constant 'title'](#)



codemanship



**TDD Tip #26:** When opponents of TDD in your organisation say it's "untried" in industry, tell them developers have been doing it successfully since the 1950s

Project Mercury ran with very short (half-day) iterations that were time boxed. The development team conducted a technical review of all changes, and, interestingly, applied the Extreme Programming practice of **test-first development**, planning and writing tests before each micro-increment. They also practiced top-down development with stubs.

Craig Larman & Victor Basili  
Iterative development on NASA's Project Mercury, 1957  
*Iterative & Incremental Development: A Brief History*



codemanship

## TDD Tip #27: Failing tests (examples) can drive the design of business operations as well as software



Jane Smith calls to complain that her order (no. 10739) for 6x Lenovo laptops hasn't arrived

The call centre tracks the order, and learns we only had 4 in stock. 2 are on order. Delivery is expected to be another 14 days. We offer to ship the 4 we have now, and the remaining 2 at no cost later. We also offer to throw in MS Office at no charge.



codemanship

**TDD Tip #28:** Test doubles can enable us to test asynchronous code synchronously and simplify concurrent designs

```
public class InboxTests {  
  
    @Test  
    public void sendsMessageToQueueForProcessing() {  
        MailQueue queue = mock(MailQueue.class);  
        Message message = mock(Message.class); // dummy  
        Inbox inbox = new Inbox(queue);  
        inbox.sendAsync(message);  
        verify(queue).send(message);  
    }  
  
    @Test  
    public void sentMessagesAddedToSentList() {  
        Sender sender = new Inbox(null); // Callback interface  
        Message message = mock(Message.class);  
        // callback method, invoked when queue has sent message  
        sender.messageSent(message);  
        assertThat(((Inbox) sender).sentMessages(), hasItem(message));  
    }  
}
```



codemanship

## TDD Tip #29: We can use test doubles to make tests that use changing or random data repeatable

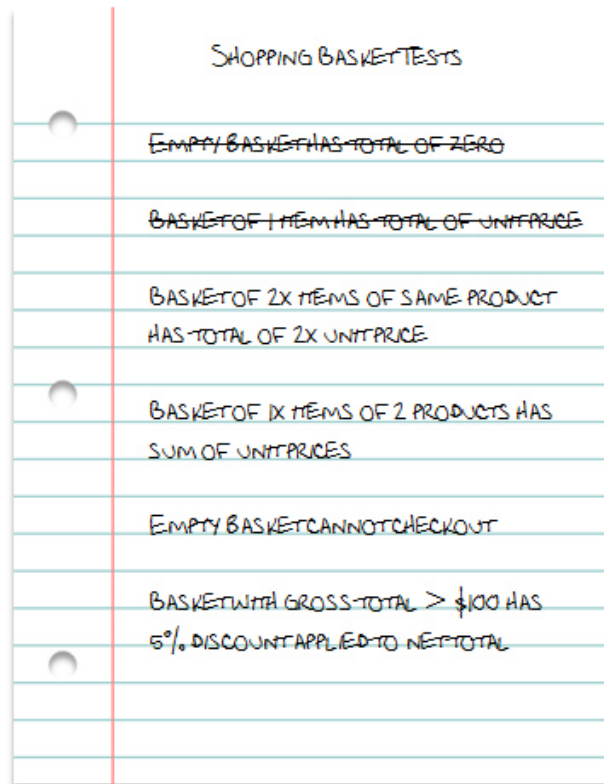
```
@Test
public void underageCustomerCannotRentEighteenRatedTitle() {
    VideoTitle title = new VideoTitle(Rating.EIGHTEEN);
    CurrentDate date = mock(CurrentDate.class);
    when(date.get()).thenReturn("31/12/2016");
    Customer customer = new Customer("1/1/1999", date);
    assertFalse(customer.canRent(title));
}
```

```
@Test
public void doubleSixGivesPlayerExtraMove() {
    Player player1 = new Player();
    Player player2 = new Player();
    Dice dice = mock(Dice.class);
    when(dice.randomThrow()).thenReturn(new int[]{6,6});
    Game game = new Game(dice, player1, player2);
    game.move(player1);
    assertThat(game.nextPlayer(), is(player1));
}
```



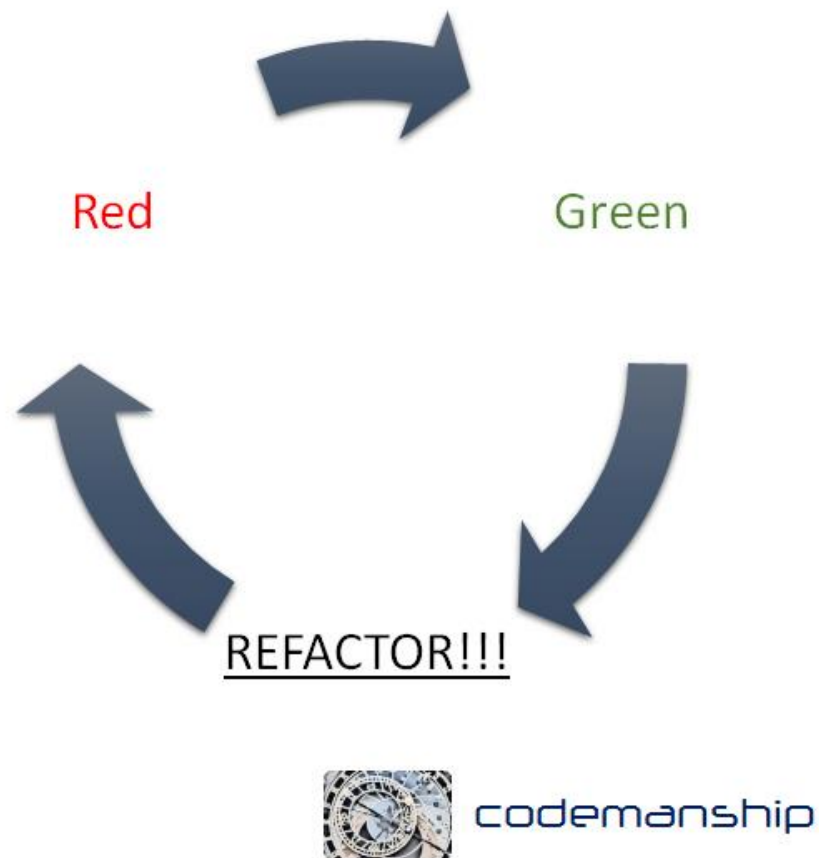
codemanship

**TDD Tip #30:** Jot down new tests you think of for working on later, so you can stay focused on the current test



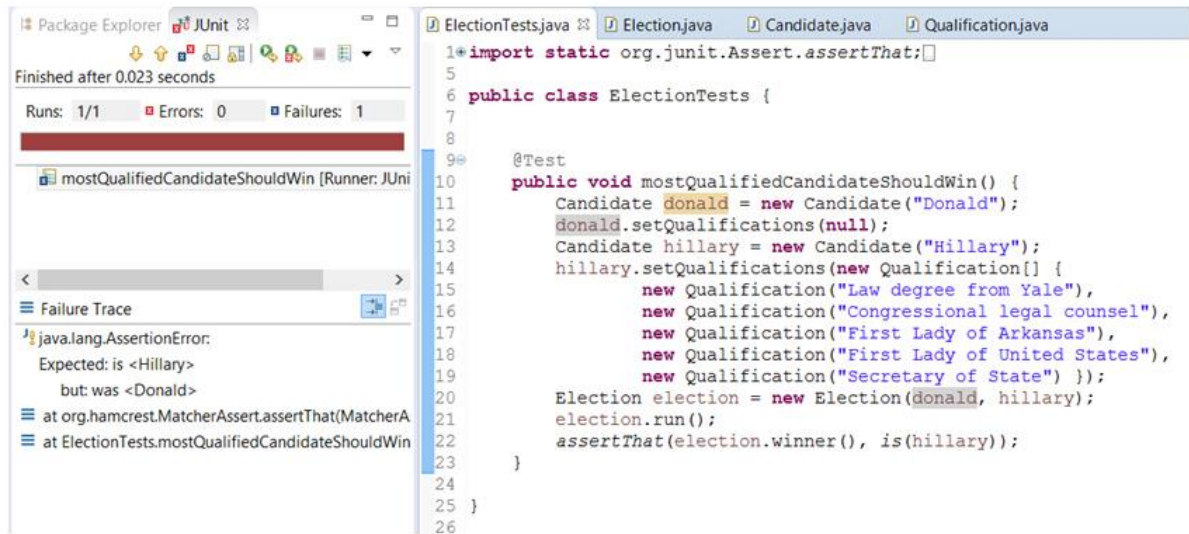
codemanship

**TDD Tip #31:** 90% of the time, developers don't go back to fix code quality issues. Don't move on to the next test unless you're happy leaving the code as it is.





**TDD Tip #32:** When users report a system failure, write a failing test before you fix it, so you can get early warning if it returns



```

1 import static org.junit.Assert.assertThat;
2
3 public class ElectionTests {
4
5     @Test
6     public void mostQualifiedCandidateShouldWin() {
7         Candidate donald = new Candidate("Donald");
8         donald.setQualifications(null);
9         Candidate hillary = new Candidate("Hillary");
10        hillary.setQualifications(new Qualification[] {
11            new Qualification("Law degree from Yale"),
12            new Qualification("Congressional legal counsel"),
13            new Qualification("First Lady of Arkansas"),
14            new Qualification("First Lady of United States"),
15            new Qualification("Secretary of State") });
16        Election election = new Election(donald, hillary);
17        election.run();
18        assertThat(election.winner(), is(hillary));
19    }
20 }
  
```



codemanship

**TDD Tip #33:** Contract tests can serve as a specification for components other developers might be working on

```

public class CommentTests {
    private final String sanitizedText = "You are a *** ** head!";

    @Test
    public void commentTextIsSanitized() {
        WebComment comment = createComment();
        comment.setText("You are a poo poo head!");
        assertEquals(sanitizedText, comment.getText());
    }

    protected WebComment createComment() {
        CommentSanitizer sanitizer = mock(CommentSanitizer.class);
        when(sanitizer.sanitize(anyString())).thenReturn(sanitizedText);
        return new WebComment(sanitizer);
    }
}
  
```

Team A works on implementing

```

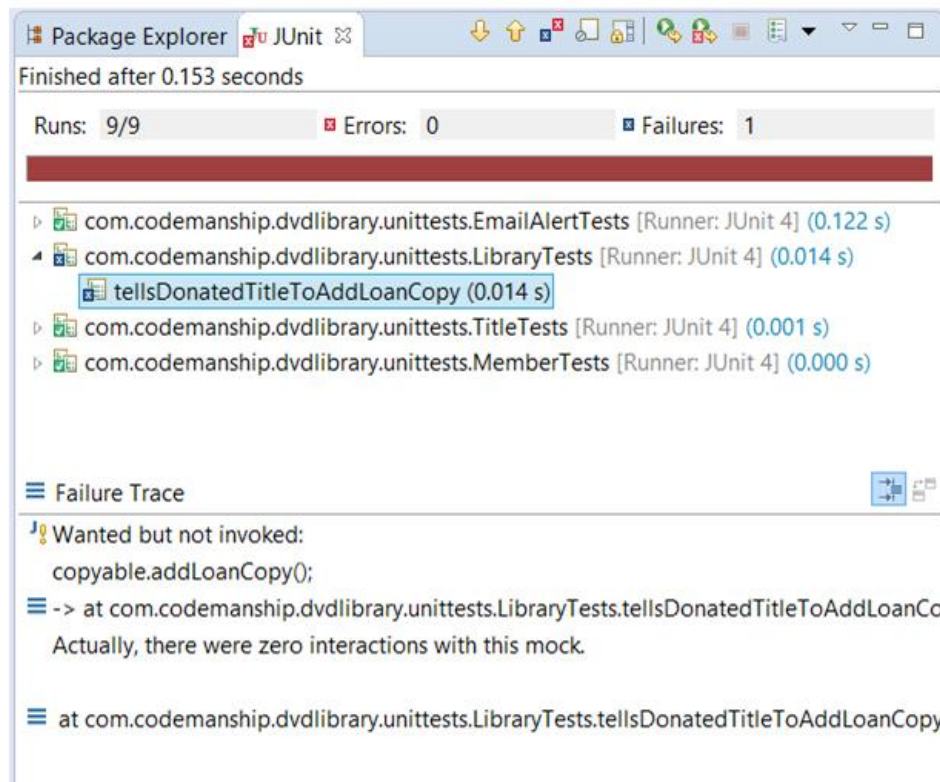
public class CommentIntegrationTests extends CommentTests {
    @Override
    protected WebComment createComment() {
        return new WebComment(new SuperConservativeCommentSanitizer());
    }
}
  
```

Team B works on implementing



codemanship

**TDD Tip #34:** Leaving a test failing when you step away from the code (e.g., for lunch) can help you quickly “find your place” again when you return



codemanship

**TDD Tip #35:** Mock objects are an easy way to implement the Null Object pattern for test dummies you know methods will be invoked on

```
public class BankTransfer implements Transaction {

    private double amount;
    private Account payer;
    private Account payee;
    private AuditTrail audit;

    public BankTransfer(double amount,
                        Account payer,
                        Account payee,
                        AuditTrail audit) {
        this.amount = amount;
        this.payer = payer;
        this.payee = payee;
        this.audit = audit;
    }

    public void execute() {
        payer.debit(amount);
        payee.credit(amount);
        audit.log(this);
    }
}

@Test
public void transferAmountDebitedFromPayer() {
    Account payer = new Account();
    payer.credit(100);
    Account payee = new Account();
    BankTransfer transfer = new BankTransfer(
        50,
        payer,
        payee,
        mock(AuditTrail.class)
    );

    transfer.execute();
    assertEquals(50, payer.getBalance(), 0);
}
```



codemanship

### TDD Tip #36: Example data removes ambiguity from customer tests



"hot"

"sweet"



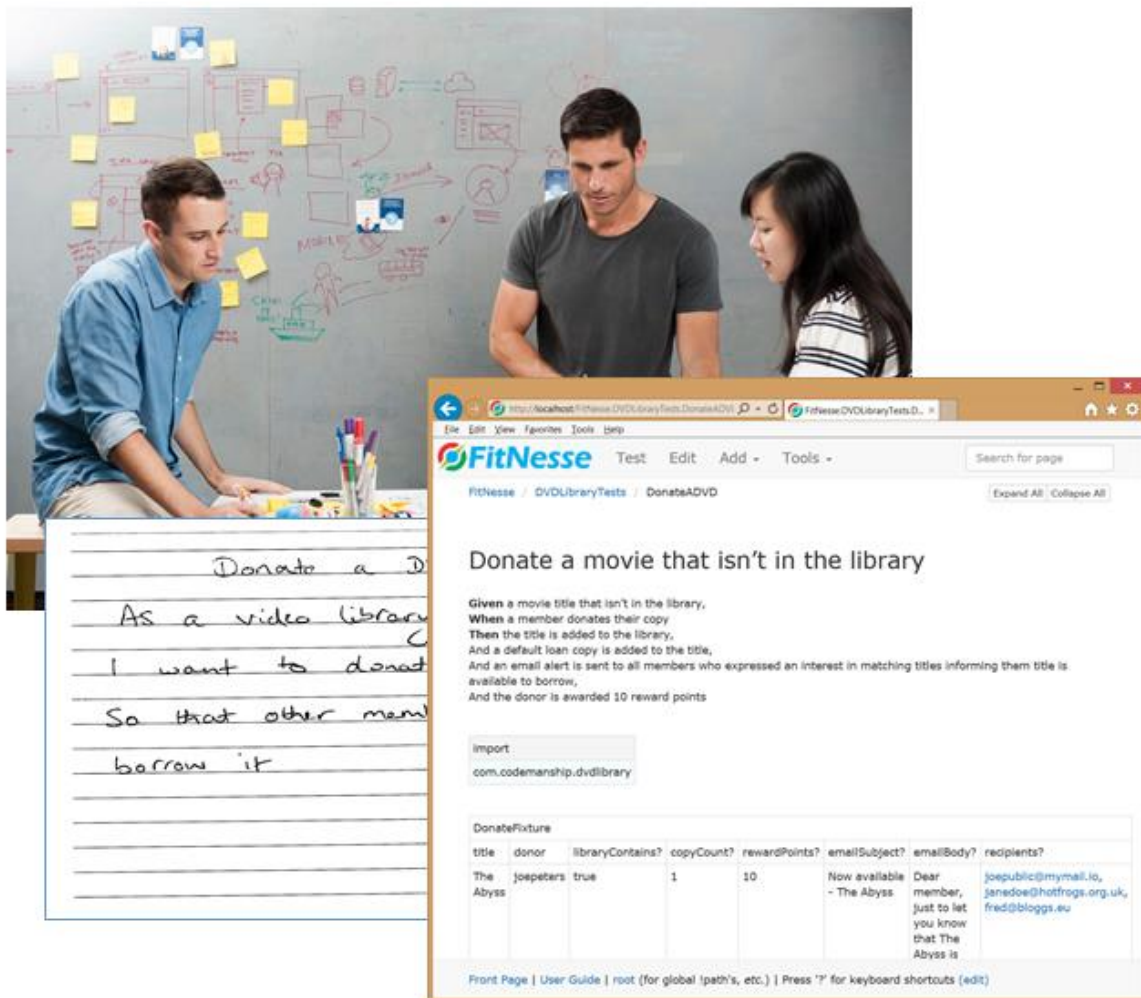
"hot" = 90°C

"sweet" = 40g/L sugar



codemanship

## TDD Tip #37: In a test-driven approach to design, testers are requirements analysts



**Donate a movie that isn't in the library**

Given a movie title that isn't in the library,  
 When a member donates their copy  
 Then the title is added to the library,  
 And a default loan copy is added to the title,  
 And an email alert is sent to all members who expressed an interest in matching titles informing them title is available to borrow,  
 And the donor is awarded 10 reward points

Import  
 com.codemanship.dvdlibrary

DonateFixture							
title	donor	libraryContains?	copyCount?	rewardPoints?	emailSubject?	emailBody?	recipients?
The Abyss	joepeters	true	1	10	Now available - The Abyss	Dear member, just to let you know that The Abyss is	joepublic@mymail.io, janedoe@hotfrogs.org.uk, fred@blogs.eu

Front Page | User Guide | root (for global lpath's, etc.) | Press '?' for keyboard shortcuts (edit)



codemanship



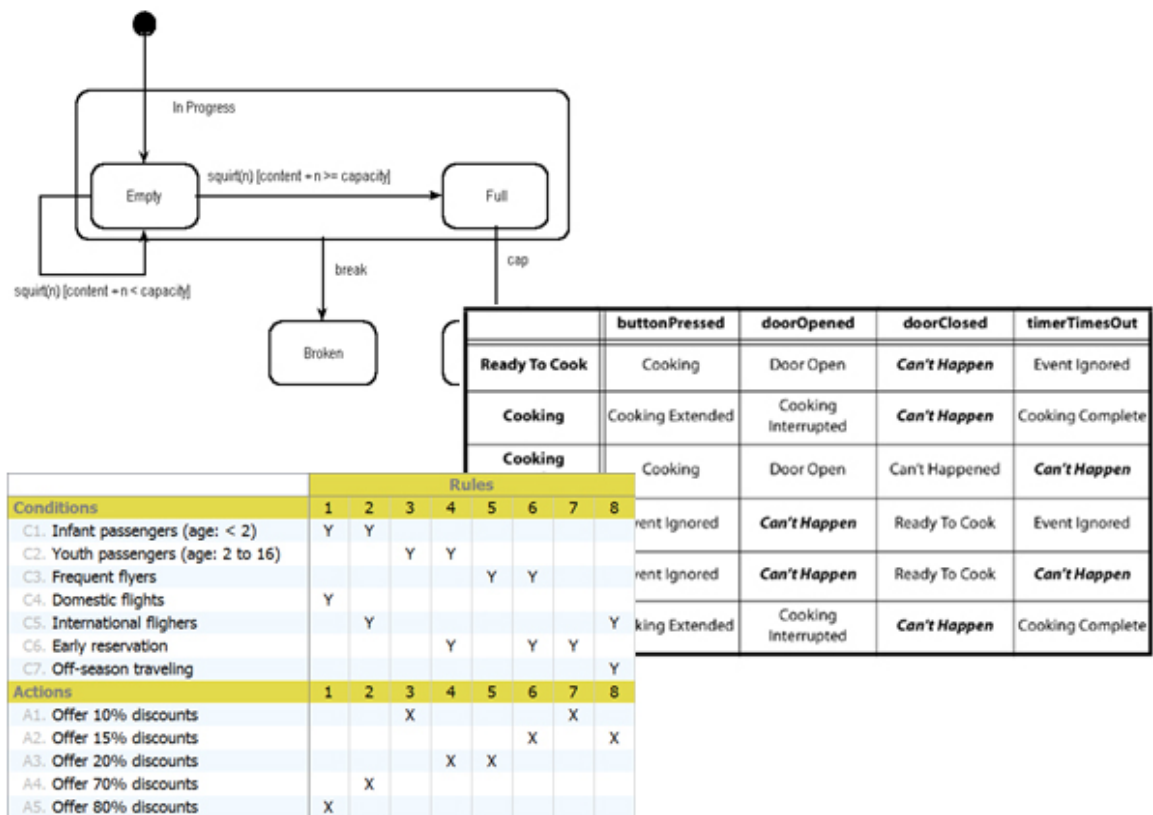
**TDD Tip #38:** Avoiding mistakes is cheaper than fixing them.  
Slow down to go faster!



codemanship



## TDD Tip #39: For complex logic, use models to visualise and help spot tests you might otherwise miss



codemanship

**TDD Tip #40:** Check for backwards compatibility by running older versions of API tests against new implementations

```
public class WarehouseApiTests {  
  
    private WarehouseApi warehouse;  
  
    @Before  
    public void createWarehouse() {  
        warehouse = new Warehouse("http://localhost/warehouse");  
    }  
  
    @Test  
    public void checkProductStock() {  
        assertEquals(1, warehouse.checkStock("GI Joe"));  
    }  
  
    @Test  
    public void orderIsShippedToGivenAddress() {  
        Address address = new Address("10 Acacia Lane, Trumpton, TRU9 3RT");  
        Order order = new Order("GI Joe", 1, address );  
        ShippingNote shipping = warehouse.fulfil(order);  
        assertEquals(address, shipping.getAddress());  
    }  
}
```



codemanship

**TDD Tip #41:** Under delivery pressure, TDD newbies revert to old habits. With regular practice, build up to applying it to everyday work.

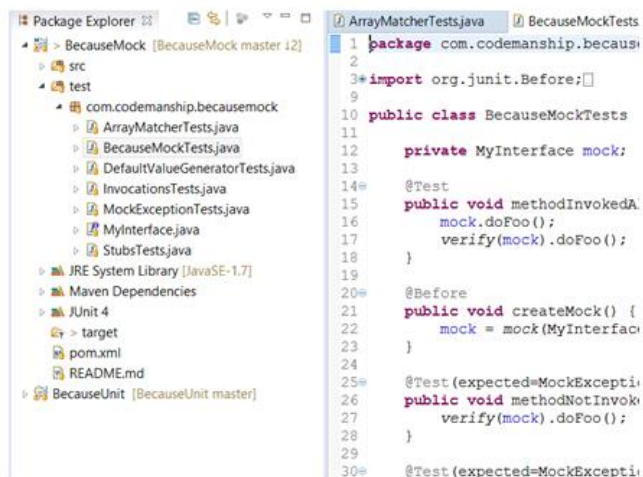
```
@Test
public void numbersDivisibleByThreeReplacedWithFizz() {
    assertEquals("Fizz", new FizzBuzz().fizzBuzz(3));
}

@Test
public void firstFibonacciNumberIsZero() {
    assertEquals(0, new Fibonacci().get(0));
}

@Test
public void romanNumeralForFourIsIV() {
    assertEquals("IV", new RomanNumerals().convert(4));
}
```

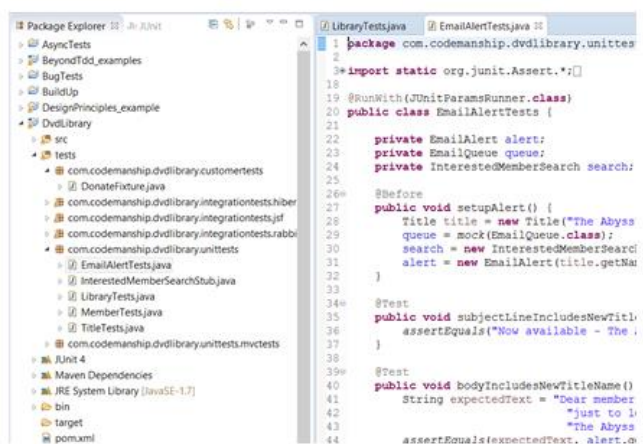
Week 1-6 : basic TDD exercises

~2 hours per week



Week 7-12 : small self-contained projects

~3-4 hours per week



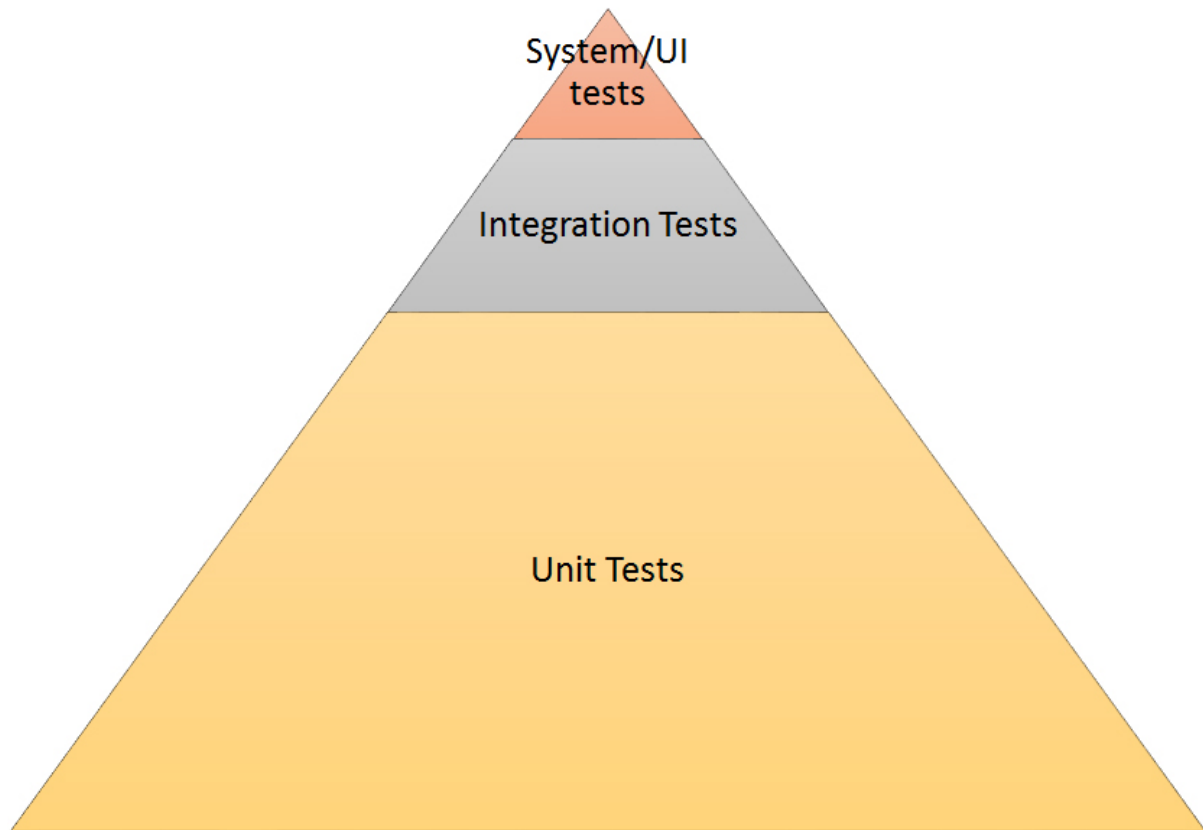
Week 13 - 24 : apply TDD to real projects, starting at a comfortable pace when less delivery pressure

~8-40 hours per week



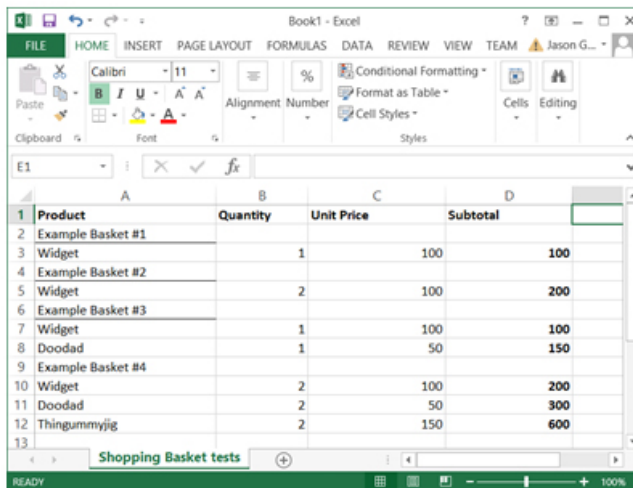
codemanship

**TDD Tip #42:** Favour fast-running unit tests so you can re-test as much logic as possible quickly for more frequent feedback



**codemanship**

## TDD Tip #43: Don't expect your customer to learn new tools so they can participate in defining tests

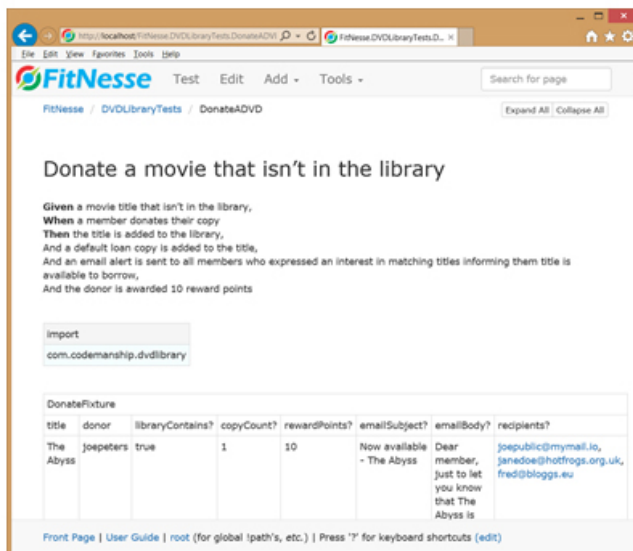


Book1 - Excel

Product	Quantity	Unit Price	Subtotal
Example Basket #1			
Widget	1	100	100
Example Basket #2			
Widget	2	100	200
Example Basket #3			
Widget	1	100	100
Doodad	1	50	150
Example Basket #4			
Widget	2	100	200
Doodad	2	50	300
Thingummyjig	2	150	600

Shopping Basket tests

Let them use familiar tools you can easily extract test data from



FitNesse Test Edit Add Tools

FitNesse / DVDLibraryTests / DonateADVD

### Donate a movie that isn't in the library

Given a movie title that isn't in the library,  
 When a member donates their copy  
 Then the title is added to the library,  
 And a default loan copy is added to the title,  
 And an email alert is sent to all members who expressed an interest in matching titles informing them title is available to borrow,  
 And the donor is awarded 10 reward points

Import  
 com.codemanship.dvdlibrary

title	donor	libraryContains?	copyCount?	rewardPoints?	emailSubject?	emailBody?	recipients?
The Abyss	joepeters	true	1	10	Now available - The Abyss	Dear member, just to let you know that The Abyss is	joepublic@mymail.io, janedoe@hottrogs.org.uk, fred@blogs.eu

Front Page | User Guide | root (for global !path's, etc.) | Press '?' for keyboard shortcuts (edit)

Drive unfamiliar ATDD/BDD tools for them during collaborative sessions



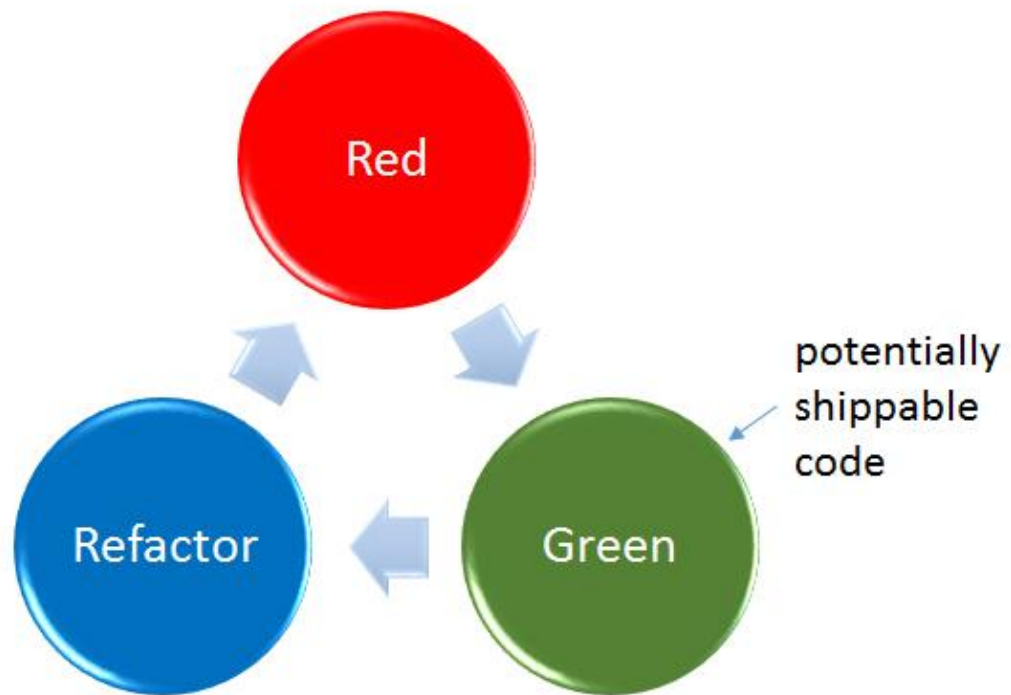
Design tools to work the way they think (which nobody's done yet!)



codemanship



**TDD Tip #44:** TDD is a gateway drug to Continuous Delivery



codemanship

**TDD Tip #45:** The opposite of duplication is reuse. Refactoring duplicate code during TDD often reveals useful abstractions.

```
public class FatherJackCensor {

    // censors Father Ted scripts for more conservative tastes
    public String sanitize(String text) {
        text = text.replaceAll("feck", "*****");
        text = text.replaceAll("drink", "*****");
        text = text.replaceAll("girls", "*****");
        return text;
    }

}
```



```
public class FatherJackCensor {

    private String[] profaneWords;

    public FatherJackCensor(String[] profaneWords){
        this.profanWords = profaneWords;
    }

    // censors Father Ted scripts for more conservative tastes
    public String sanitize(String text) {
        for (String word : profaneWords) {
            text = replaceWord(text, word);
        }
        return text;
    }

    private String replaceWord(String text, String word) {
        return text.replaceAll(word, word.replaceAll(".", "*"));
    }

}
```



codemanship

**TDD Tip #46: “Adversarial” pairing can help you write better tests**

Fred writes a failing test

```
@Test
public void borrowedTitlesAreAddedToMembersLoans() {
    Member member = new Member();
    VideoTitle title = new VideoTitle();
    member.borrow(title);
    assertEquals(1, member.getLoans().size());
}
```



Emma writes code that passes the test, but isn't what Fred intended

```
public class Member {

    public List<VideoTitle> getLoans() {
        return Arrays.asList(new VideoTitle[]{null});
    }

    public void borrow(VideoTitle title) {
    }

}
```



Fred makes the test stronger

```
@Test
public void borrowedTitlesAreAddedToMembersLoans() {
    Member member = new Member();
    VideoTitle title = new VideoTitle();
    member.borrow(title);
    assertThat(member.getLoans(), contains(title));
}
```



codemanship

**TDD Tip #47:** In TDD, we don't have to build features in a "logical order". Using test doubles, we can fake it until we make it.



Obviously, we'll need to get payment processing working before we can tackle order fulfilment



Let's *fake* payment processing using a mock object so we can work on fulfilling orders



codemanship

**TDD Tip #48:** A good way to know if you're ready to apply TDD to commercial projects is when it doesn't slow you significantly



Time yourself tackling a non-trivial problem (e.g., Roman numeral converter) without applying TDD



Time yourself tackling the same problem applying TDD *rigorously*

- Ask a colleague to acceptance test the solution thoroughly when you believe you are finished. *You aren't done until they say so.*
- Repeat the exercise 3 times so that learning from one pass is applied to the next pass. Consider the first pass to be a trial run, and only use the timings from the next 2 passes
- Also ask your colleague to review the quality of the code. *You aren't done until they say the code's clean enough.*
- If you can TDD it without taking more than ~20% longer, you're probably ready to start TDDing real production code



codemanship



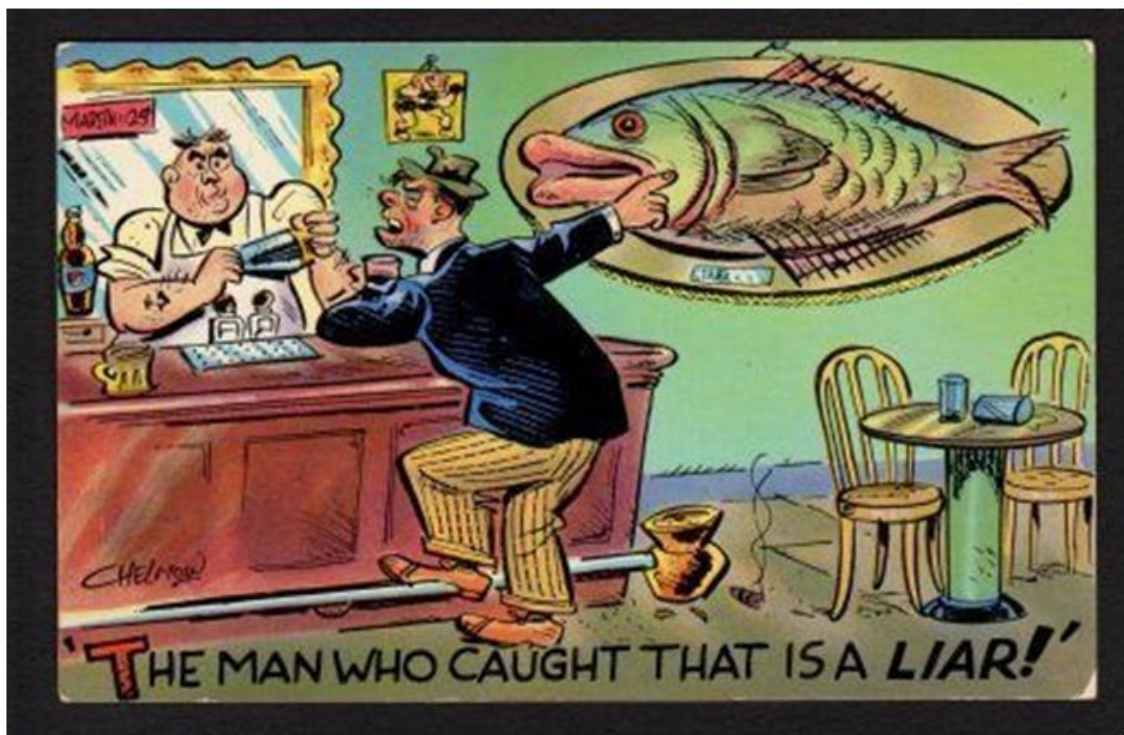
**TDD Tip #49:** Under pressure, we revert to our default way of working. Focus on building *good habits* to make TDD your default behaviour.

- Start by writing a failing test
- Write the simplest, quickest code to pass the test
- Refactor to make the next test easier
- Don't write source code unless you have a failing test that requires it
- Triangulate when the solution isn't trivial
- Don't refactor when tests are failing
- Write the assertion first and work backwards
- See the test assertion fail so you know it's a good test
- Write tests that have one reason to fail
- Write tests that clearly convey their intent
- Cleanly separate test code and source code
- Organise test code to make it easy to see what's being tested
- Write tests that can run individually and in any order



codemanship

**TDD Tip #50:** The best time to sell the benefits of TDD is after you've achieved them. Under-promise and over-deliver.



codemanship

## TDD Tip #51: You can use the Builder pattern to encapsulate duplicated set-up code

```
@Test
public void membersGetTenPercentDiscount() {
    Membership membership = new Membership("08-12-2016");
    Country country = new Country("United Kingdom");
    Address address = new Address("1 High Street", "Nontown", "NT1 5AA", country);
    Customer customer = new Customer("Bill Smith", address, membership);
    ShoppingCart cart = new ShoppingCart(customer);
    cart.add(new Item("Widget", 1, 10.0));
    assertEquals(9.0, cart.netTotal(), 0);
}

@Test
public void nonMembersGetNoDiscount() {
    Country country = new Country("United Kingdom");
    Address address = new Address("1 High Street", "Nontown", "NT1 5AA", country);
    Customer customer = new Customer("Bill Smith", address, null);
    ShoppingCart cart = new ShoppingCart(customer);
    cart.add(new Item("Widget", 1, 10.0));
    assertEquals(10.0, cart.netTotal(), 0);
}

@Test
public void noFreeShippingOutsideUK() {
    Country country = new Country("France");
    Address address = new Address("1 High Street", "Nontown", "NT1 5AA", country);
    Customer customer = new Customer("Bill Smith", address, null);
    ShoppingCart cart = new ShoppingCart(customer);
    cart.add(new Item("Widget", 1, 10.0));
    assertFalse(cart.freeShipping());
}
```



```
@Test
public void membersGetTenPercentDiscount() {
    ShoppingCart cart = ShoppingCartBuilder
        .aShoppingCart()
        .withMembership()
        .build();
    assertEquals(9.0, cart.netTotal(), 0);
}

@Test
public void nonMembersGetNoDiscount() {
    ShoppingCart cart = ShoppingCartBuilder
        .aShoppingCart()
        .build();
    assertEquals(10.0, cart.netTotal(), 0);
}

@Test
public void noFreeShippingOutsideUK() {
    ShoppingCart cart = ShoppingCartBuilder
        .aShoppingCart()
        .withCountry("France")
        .build();
    assertFalse(cart.freeShipping());
}
```



codemanship

**TDD Tip #52: Code isn't the only thing that can be test-driven**

**FitNesse / DVDLibraryTests / DonateADVD**

Donate a movie that isn't in the library

**Given** a movie title that isn't in the library,  
**When** a member donates their copy  
**Then** the title is added to the library,  
And a default loan copy is added to the title,  
And an email alert is sent to all members who expressed an interest in matching titles informing them that the title is now available to borrow,  
And the donor is awarded 10 reward points

import  
com.codemanship.dvdlibrary

title	donor	libraryContains?	copyCount?	rewardPoints?	emailSubject?	emailBody?	recipients?
The Abyss	joepeters	true	1	10	Now available - The Abyss	Dear member, just to let you know that The Abyss is	Joepublic@nynemo, Janedoe@hotfrogs.org.uk, fred@blogs.eu

Front Page | User Guide | root (for global !path's, etc.) | Press '?' for keyboard shortcuts (edit)

Welcome, joepeters!  
You have 25 reward points

**Donate**

Title	Rental Copies
Star Wars	3
Some Like It Hot	
The French Connection	


Title:

**OK** **cancel**

Welcome, joepeters!  
You have 35 reward points

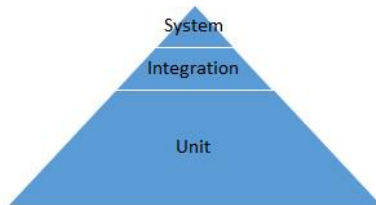
**Donate**

Title	Rental Copies
Star Wars	3
Some Like It Hot	1
The French Connection	2
The Abyss	1

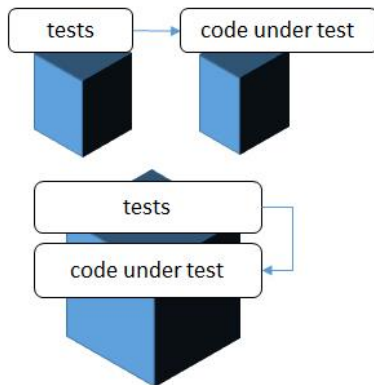
 **codemanship**



**TDD Tip #53:** Tests that run faster can be run more often. Optimise your automated test suites as they grow.

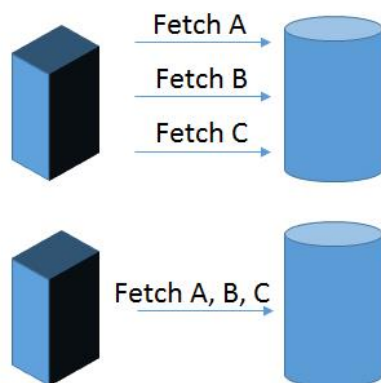


Minimise slow-running tests  
(favour fast-running unit tests)



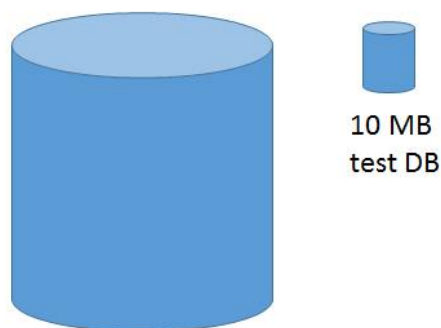
Localise test execution and  
code under test (ideally, in  
same process)

e.g., run browser tests on  
same machine as web server,  
use in-memory database, run  
JS UI tests in fake browser  
container



Batch expensive network  
communications

e.g., send all the DB queries  
for a web page with multiple  
datasets in the same SQL  
command



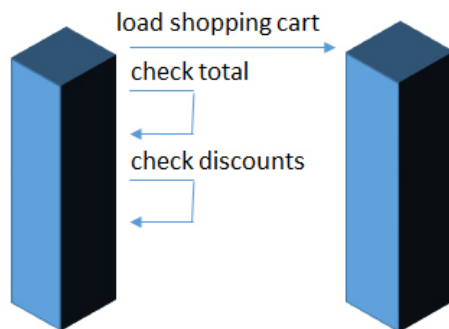
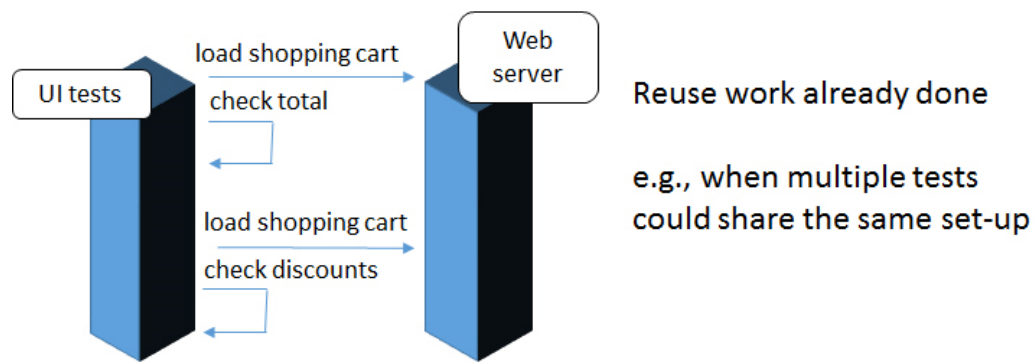
100 GB copy of live DB

10 MB  
test DB

Test with the smallest  
possible datasets



(TDD Tip #53 continued)



Intel i3 CPU  
4 GB RAM  
500 GB HDD



Intel i7 CPU  
32 GB RAM  
512 GB SSD

Upgrading hardware is often much cheaper than a developer's time



codemanship

**TDD Tip #54:** If your implementation contains conditional logic to pass the first test, you're doing TDD wrong

```
@Test
public void rootOfOneIsOne() {
    assertEquals(1, MathUtils.sqrroot(1), 0);
}

public class MathUtils {

    public static double sqrroot(double input) {
        if(input < 0) throw new IllegalArgumentException();
        return 1;
    }
}
```



codemanship

## TDD Tip #55: If your tests contain conditional logic, they may well be testing more than one thing

```
@RunWith(JUnitParamsRunner.class)
public class SquareRootTests {

    @Rule
    public ExpectedException thrown = ExpectedException.none();

    @Test
    @Parameters({"-1", "0", "0.5", "1", "4", "9"})
    public void squareOfSquareRootIsSameAsInput(double input) {
        if(input < 0)
            thrown.expect(IllegalArgumentException.class);
        double root = MathUtils.sqrroot(input);
        assertEquals(input, root * root, 0.00001);
    }
}
```



refactor

```
@RunWith(JUnitParamsRunner.class)
public class SquareRootTests {

    @Rule
    public ExpectedException thrown = ExpectedException.none();

    @Test
    @Parameters({"0", "0.5", "1", "4", "9"})
    public void squareOfSquareRootIsSameAsInput(double input) {
        double root = MathUtils.sqrroot(input);
        assertEquals(input, root * root, 0.00001);
    }

    @Test
    public void cannotSquareRootNegativeInputs() {
        thrown.expect(IllegalArgumentException.class);
        MathUtils.sqrroot(-1);
    }
}
```



codemanship

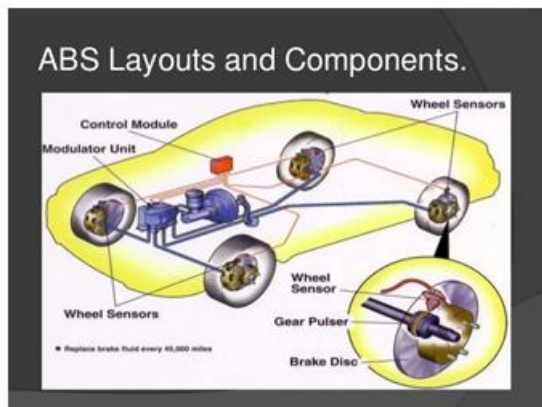
**TDD Tip #56:** If you think of a test case you missed when agreeing acceptance tests, talk to the customer before adding it to the code

<input type="radio"/>	Donate movie to library tests
	1. Donate additional copy of existing title
	2. Donate copy of new title
	3. Donate multiple copies of same title
<input type="radio"/>	4. Loan copy to library for fixed term
	5. Donate copies in different formats (DVD, VHS, Blu-ray) ?



codemanship

## TDD Tip #57: Apply more rigour to test-driving critical code



Code where  
consequences of failure  
are more severe

**In stock.**

Dispatched from and sold by Amazon.

Gift-wrap available.

Quantity  Add to Shopping Basket

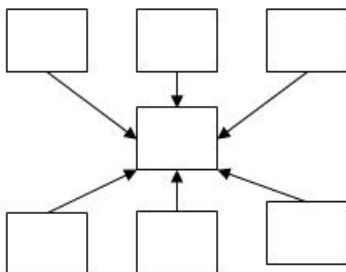


Add to Basket

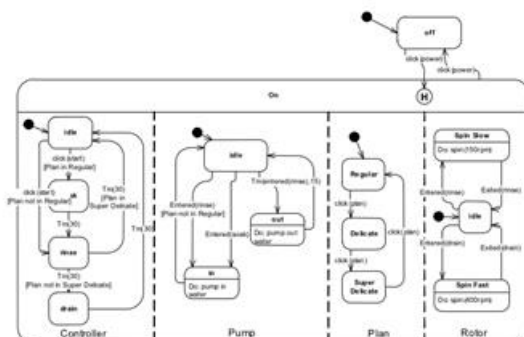
[Turn on 1-Click ordering for this browser](#)

Dispatch to:

Code that's executed  
more frequently



Code that's more  
depended upon



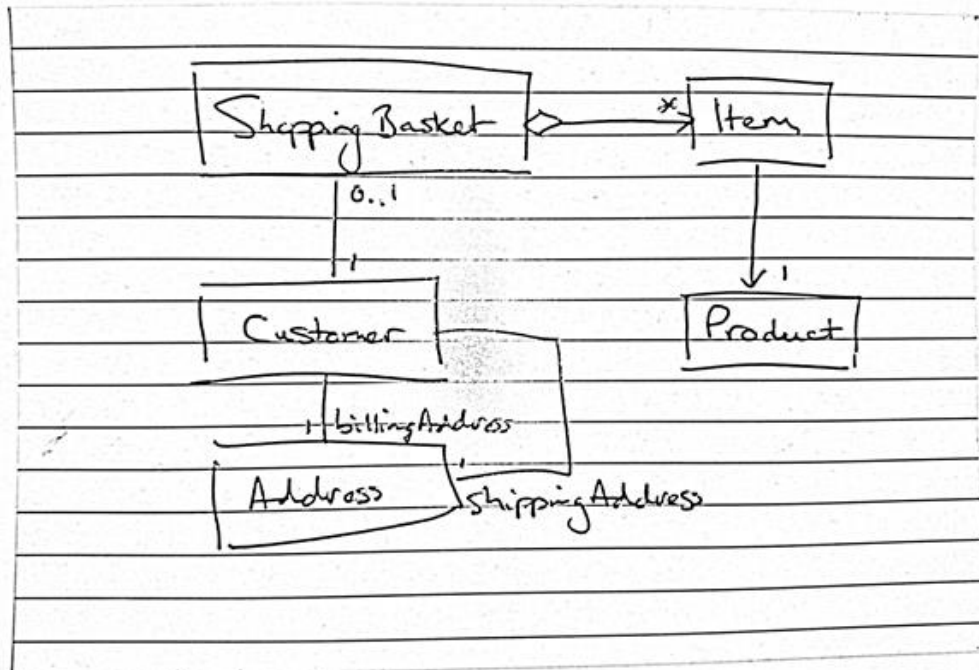
Code that has more ways  
of being wrong



codemanship

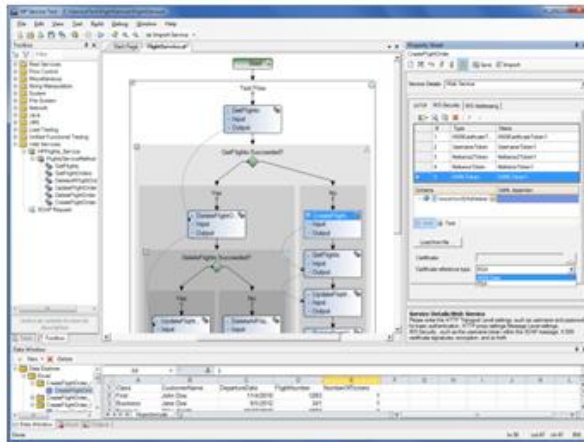


**TDD Tip #58:** Contrary to what some critics of TDD claim, it's okay to think ahead about design. In fact, we recommend it.



codemanship

## TDD Tip #59: Open Source customer testing tools make it easier to involve the whole team, and scale up/out test execution



Acme Inc. automated their customer tests using a proprietary test automation tool costing \$thousands per license. Their full regression test suite takes 4 hours to run. They can't afford more licenses.



FizzBuzz Ltd automated all their customer tests with an Open Source framework. In total, they take 4 hours to run, but they use a cloud solution to run them in parallel in < 5 minutes



codemanship

**TDD Tip #60:** The most important quality of software is that it works. While TDD-ing internal design, revisit your customer tests continuously.

DonateFixture					
title	donor	libraryContains?	copyCount?	rewardPoints?	emailSubject
The Abyss	joepeters	true	1	10	[null] expected available - Th

Answers.feature ✕

```

Feature: Ordering answers

Scenario: The answer with the highest vote gets to the top
  Given there is a question "What's your favorite colour?" with the answers
    | Answer      | Vote |
    | Red          | 1    |
    | Cucumber green | 1    |
  When you upvote answer "Cucumber green"
  Then the answer "Cucumber green" should be on top
  
```

```

feature "Signing in" do
  background do
    User.make(:email => 'user@example.com', :password => 'caplin')
  end

  scenario "Signing in with correct credentials" do
    visit '/sessions/new'
    within("#session") do
      fill_in 'Login', :with => 'user@example.com'
      fill_in 'Password', :with => 'caplin'
    end
    click_link 'Sign in'
    expect(page).to have_content 'Success'
  end
end
  
```



codemanship

**TDD Tip #61:** Effective, sustainable TDD requires us to master a range of development disciplines. It's not just about unit tests!

DonateFixture					
title	donor	libraryContains?	copyCount?	rewardPoints?	emailSubject
The Abyss	joepeters	true	1	10	[null] expected available - T

**Requirements:** we have to work with our customer to agree failing tests

Library		Title	
• Knows about titles	• Title	• Knows its name, director & year of release	
• Knows about new titles	• Member	• Knows about rental copies	
• Adds donated titles		• Registers rental copy	
• Adds new titles			

**Design:** we have to think of how our software can be composed of functions and modules that will pass the customer's tests

EmailAlert		Member	
• Sends email to members who specified matching title	?	• Knows about priority points	
		• Awards priority points	

```
@Test
public void donatedTitlesAreAddedToAvailableTitles() {
    Library library = new Library();
    Title title = mock(Title.class);
    library.donate(title);
    assertTrue(library.contains(title));
}
```

**Programming:** we have to implement our designs reliably in code

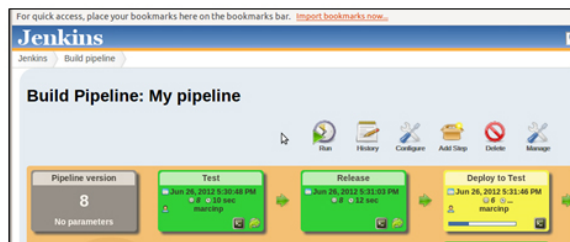
```
public class Title {
    private i
    private S

    public Ti
    this.
}
```

Refactoring options shown:

- Rename... (Alt+Shift+R)
- Move... (Alt+Shift+V)
- Extract Interface...
- Extract Superclass...
- Use Supertype Where Possible...
- Pull Up...
- Push Down...

**Refactoring:** we have to continuously apply good design principles to keep our code easy to change



**Continuous Integration:** we have to continuously merge our changes, making sure the software always works using our automated tests

condition							
Title is already in library	T	T	T	F	F	F	F
Title release date < 1 year ago	T	T	F	F	T	T	F
Title IMDB rating > 8	T	F	T	F	T	F	T
action							
add title to library					X	X	X
add loan copy to title	X	X	X	X	X	X	X
award donor 10 ADDED TITLE points					X	X	X
award donor 5 NEW RELEASE points	X	X			X	X	
award donor 5 TOP TITLE points	X		X		X		X
award donor 5 NEW COPY points	X	X	X	X			
send ADDED TITLE email alert					X	X	X

**Testing:** we have to think like testers to help our customers spot examples we might have otherwise missed



codemanship

**TDD Tip #62:** Avoid noise words like 'should' in test names. They're redundant and create clutter.

```
@Test
public void under100ShouldGetNoDiscount() {
    assertEquals(0, new Discount().calculate(99.99), 0);
}

@Test
public void from100To200ShouldGet5Percent() {
    assertEquals(5, new Discount().calculate(100.00), 0);
}

@Test
public void over200ShouldGet10Percent() {
    assertEquals(20.01, new Discount().calculate(200.10), 0);
}
```



```
@Test
public void under100GetsNoDiscount() {
    assertEquals(0, new Discount().calculate(99.99), 0);
}

@Test
public void from100To200Gets5Percent() {
    assertEquals(5, new Discount().calculate(100.00), 0);
}

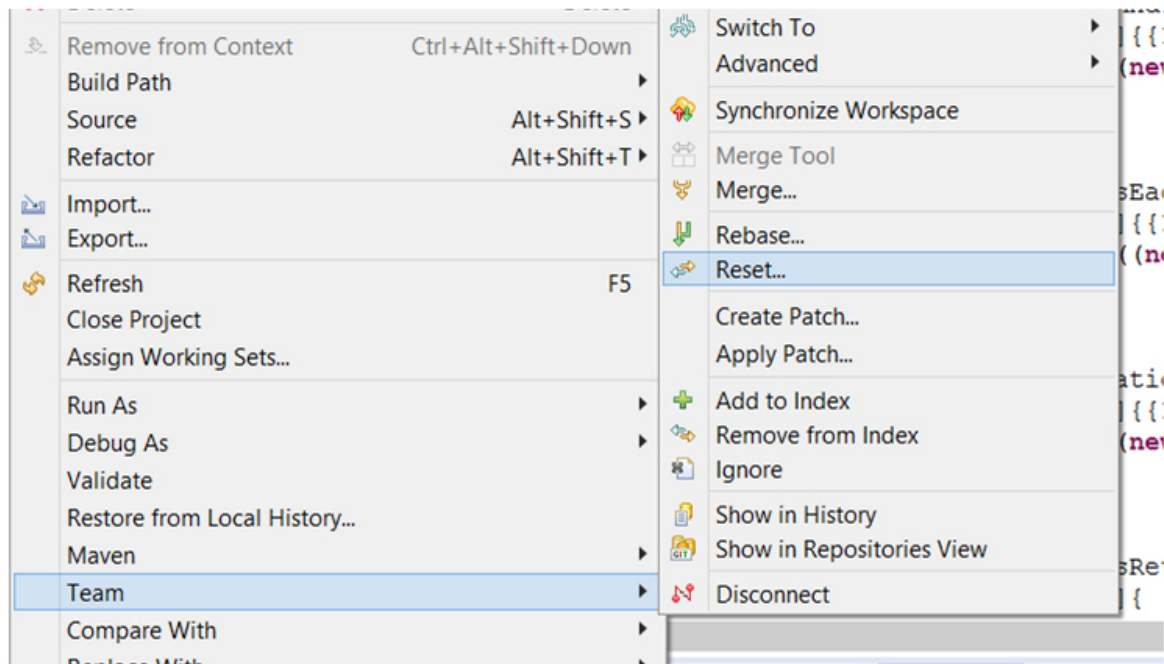
@Test
public void over200Gets10Percent() {
    assertEquals(20.01, new Discount().calculate(200.10), 0);
}
```



codemanship

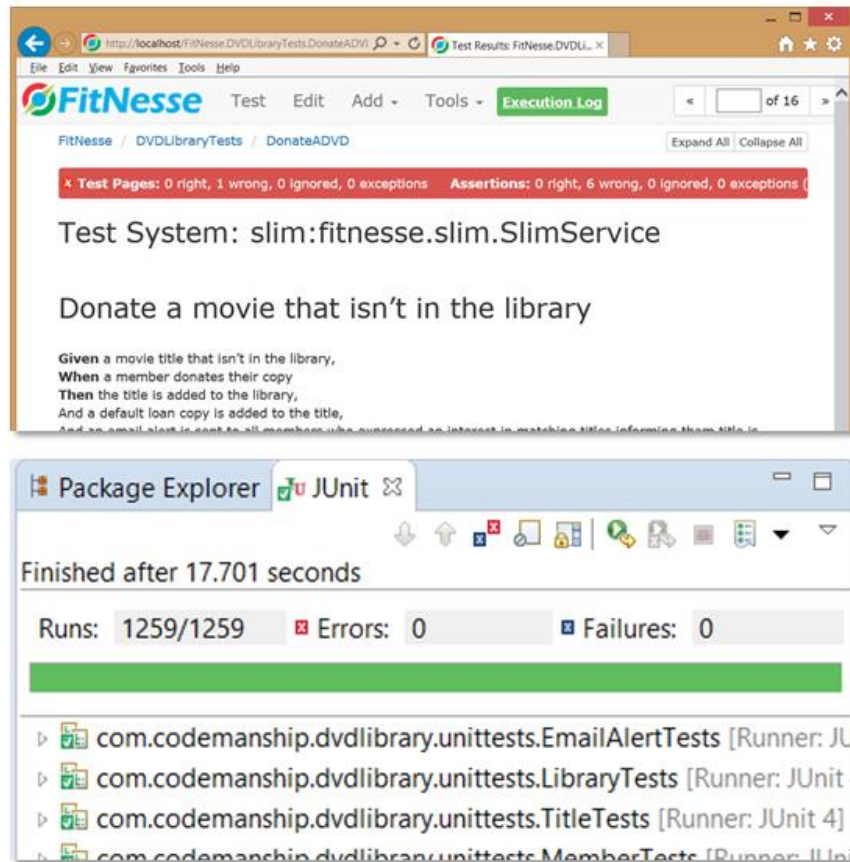


**TDD Tip #63:** Starting over is sometimes cheaper than fixing a mess, and often cheaper than living with one.



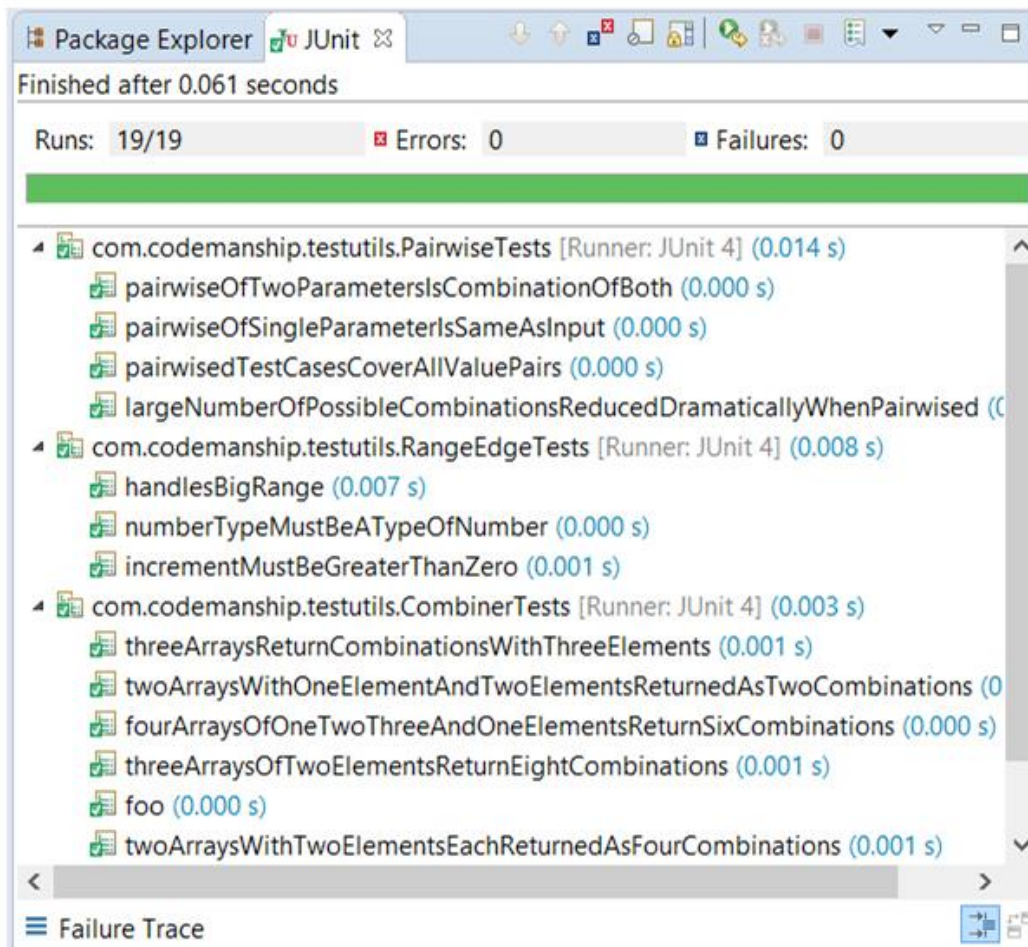
codemanship

**TDD Tip #64:** If a customer test fails, but all your developer tests pass, there's a gap in your developer tests.



codemanship

**TDD Tip #65:** When you're about to work on an existing code base for the first time, start by making sure all the tests pass



codemanship

## TDD Tip #66: Test doubles that return test doubles could be a sign that the class you're testing breaks the Law of Demeter

```
@Test
public void shippingCostsExtraPoundOutsideEU() {
    Customer customer = mock(Customer.class);
    Address address = mock(Address.class);
    Country country = mock(Country.class);
    when(customer.getAddress()).thenReturn(address);
    when(address.getCountry()).thenReturn(country);
    when(country.isInEu()).thenReturn(false);
    double euShipping = 4.95;
    Invoice invoice = new Invoice(customer, euShipping);
    assertEquals(euShipping + 1, invoice.getTotalShipping(), 0);
}
```

```
public class Invoice {

    private final Customer customer;
    private final double shipping;

    public Invoice(Customer customer, double shipping) {
        this.customer = customer;
        this.shipping = shipping;
        if(!customer.getAddress().getCountry().isInEu())
            shipping += 1.0;
    }
}
```

refactored

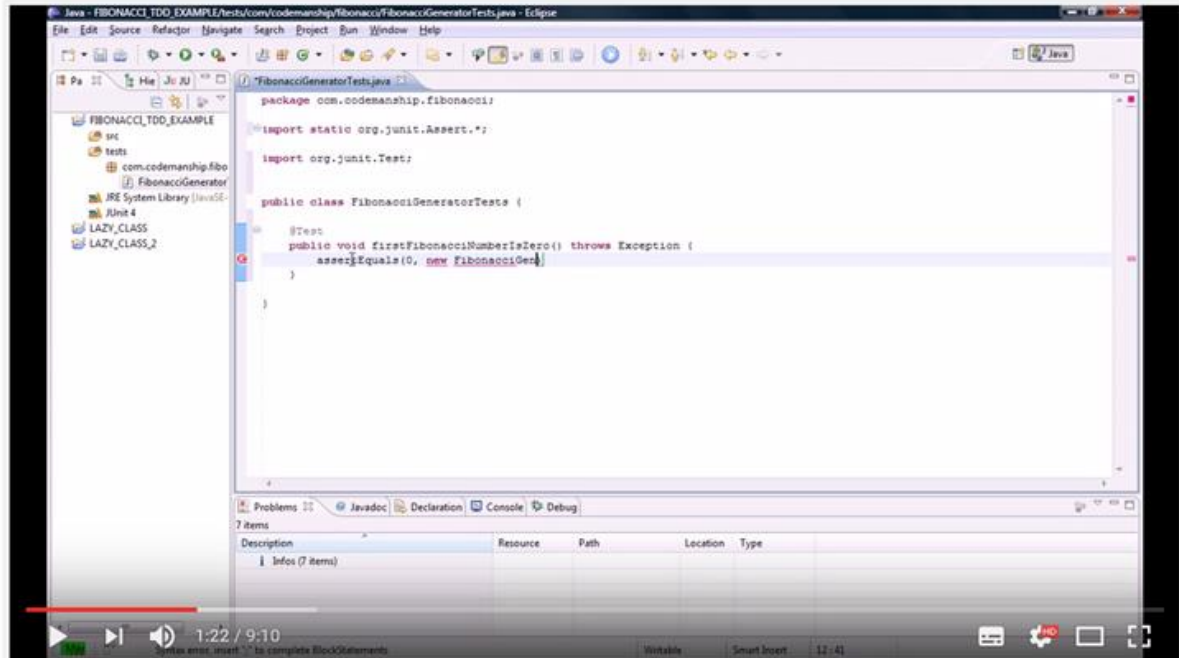
```
@Test
public void shippingCostsExtraPoundOutsideEU() {
    Customer customer = mock(Customer.class);
    when(customer.isInEu()).thenReturn(false);
    double euShipping = 4.95;
    Invoice invoice = new Invoice(customer, euShipping);
    assertEquals(euShipping + 1, invoice.getTotalShipping(), 0);
}
```



codemanship



**TDD Tip #67:** Like ballet dancers practicing in front of a mirror, it can help to watch yourself do TDD to see where you could improve



codemanship





**Codemanship**  
@codemanship

## TDD Tip #68: There's still a place for a bit of up-front analysis & design in TDD [#101TddTips](#)



**jasongorman** @jasongorman

When I'm not sure how to approach a problem, I make lists and draw little pictures. Often, a door opens #tdd

11:07 PM - 11 Jan 2017

**TDD Tip #69:** The most common reason for failing with TDD is not putting enough effort into refactoring the test code



codemanship

**TDD Tip #70:** TDD and Formal Methods have much in common.  
Experience of both can improve you at either.

```
public class HtlRoomsTest {

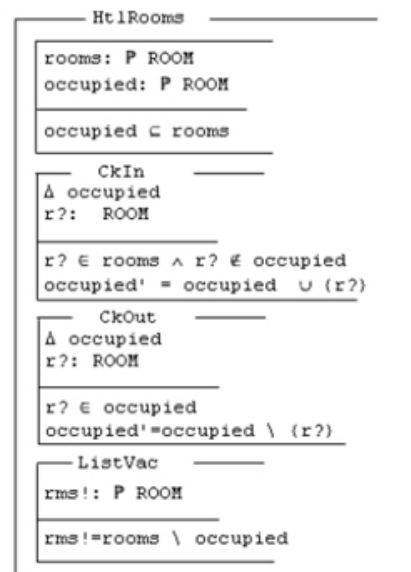
    private HtlRooms htlRooms;
    private Room room;

    @Test
    public void afterCheckInRoomIsOccupied() {
        checkIntoRoom();
        assertThat(htlRooms.getOccupied(), contains(room));
    }

    @Test(expected=RoomNotInHotelException.class)
    public void cannotCheckIntoRoomNotInHotel() {
        Room room = new Room();
        HtlRooms htlRooms = new HtlRooms(new ArrayList<Room>());
        htlRooms.ckIn(room);
    }

    @Test
    public void afterCheckoutRoomIsUnoccupied() {
        checkIntoRoom();
        htlRooms.ckOut(room);
        assertThat(htlRooms.getOccupied(), not(contains(room)));
    }

    private void checkIntoRoom() {
        room = new Room();
        List<Room> rooms = new ArrayList<>();
        rooms.add(room);
        htlRooms = new HtlRooms(rooms);
    }
}
```



codemanship

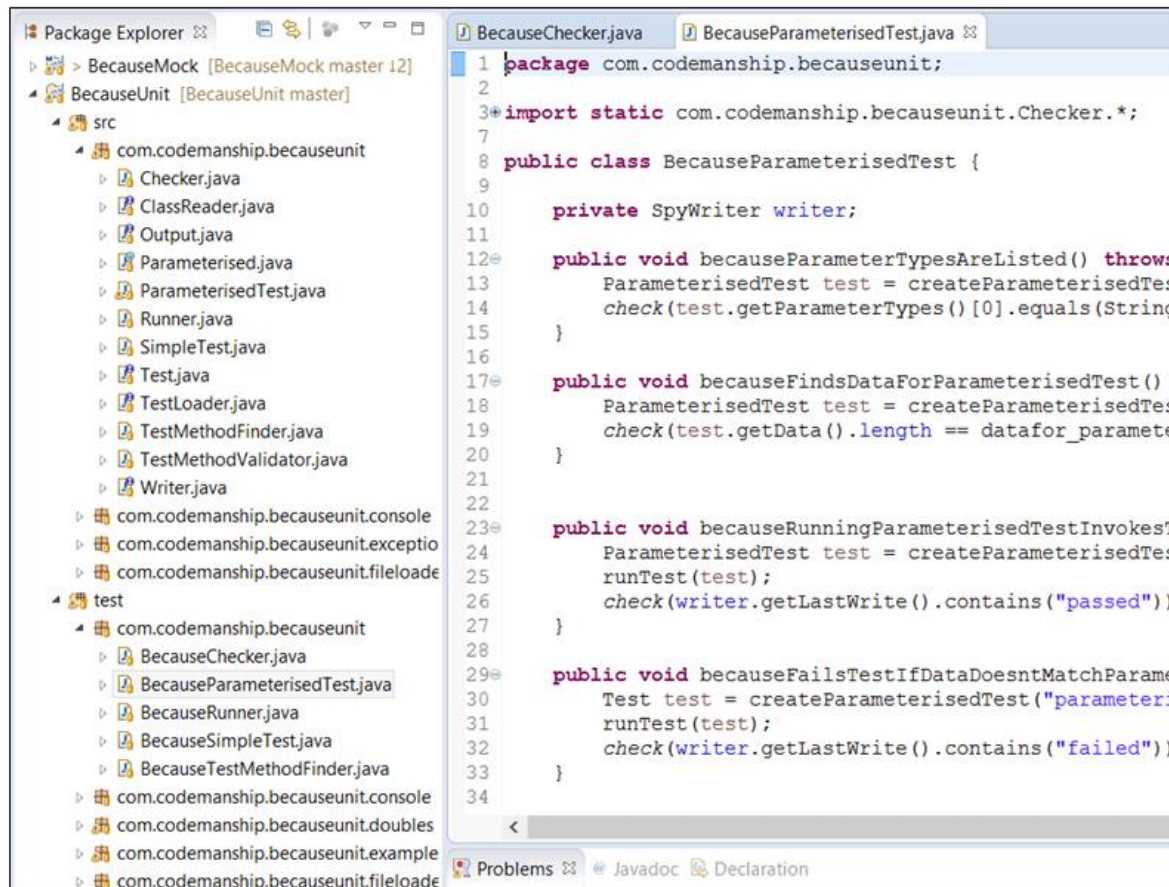
## TDD Tip #71: With a little extra code, unit tests can be reused as integration tests

```
public class ShoppingCartTests {  
  
    protected ShoppingCartBuilder builder;  
  
    @Before  
    public void setUp() {  
        builder = new ShoppingCartBuilder();  
    }  
  
    @Test  
    public void membersGetTenPercentDiscount() {  
        ShoppingCart cart = builder  
            .aShoppingCart()  
            .withMembership()  
            .build();  
        assertEquals(9.0, cart.netTotal(), 0);  
    }  
}  
  
public class ShoppingCartPayPalTests extends ShoppingCartTests {  
  
    @Before  
    @Override public void setUp() {  
        builder = new PayPalShoppingCartBuilder();  
    }  
}
```



codemanship

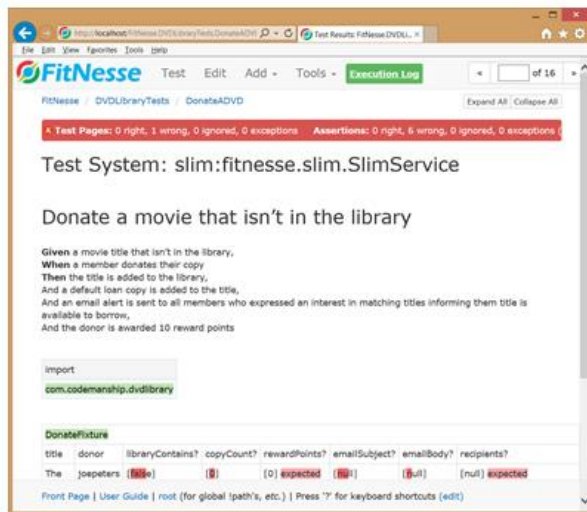
**TDD Tip #72:** Rolling your own unit testing frameworks can be a. great practice, and b. help you better appreciate their design



codemanship



**TDD Tip #73:** You can't automate customer acceptance testing. The customer needs to see it working for themselves.



Automating customer tests helps guide us to a working solution design, and can provide cheaper regression testing

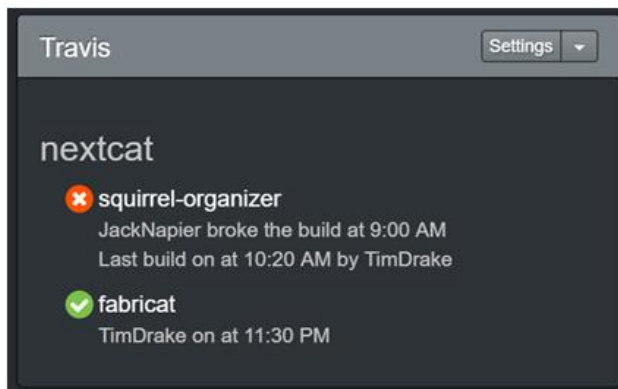


But real customer feedback from using the software is required before we can be sure we delivered what they were expecting. When it's ready, get them to execute the tests you agreed for themselves.

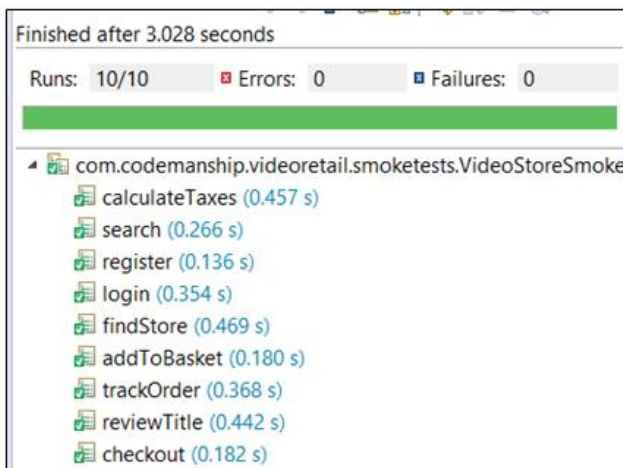


codemanship

**TDD Tip #74:** If you're the only person on your team doing TDD, be careful not to end up fixing everyone else's broken commits



Even if it's using your own personal test suite & CI server, make sure every commit gets tested. Report problems *immediately*.



Write some high-level 'smoke tests' to quickly sanity-check each commit, including code you're not working on. Just a few key user journeys could reveal obvious problems.



Don't let management bury their heads in the sand about this issue. Talk about it openly and constructively. Don't just moan. Demonstrate workable solutions and sell the benefits.

And if the team simply don't want to know, then maybe you're on the wrong team?



codemanship

**TDD Tip #75:** Avoid setting team targets for TDD like test coverage. They're too easily gamed. Pairing is the best way to see what developers do.

```
@Test
public void testEveryMethod() {
    List<Class> classes = ClassUtils.loadAllClasses();
    for (Class c : classes) {
        Object instance = ClassUtils.createInstance(c);
        Method[] methods = c.getMethods();
        for (Method m : methods) {
            Object[] defaultValues
                = ClassUtils.createDefaultParamValues(m);
            try {
                m.invoke(instance, defaultValues);
            } catch (Exception e) {
                // ignore unhandled exceptions
            }
            assertTrue(true);
        }
    }
}
```



codemanship

**TDD Tip #76:** 'Happy path' test scenarios tend to have the most value, because the end user achieves their goal. Prioritise accordingly.

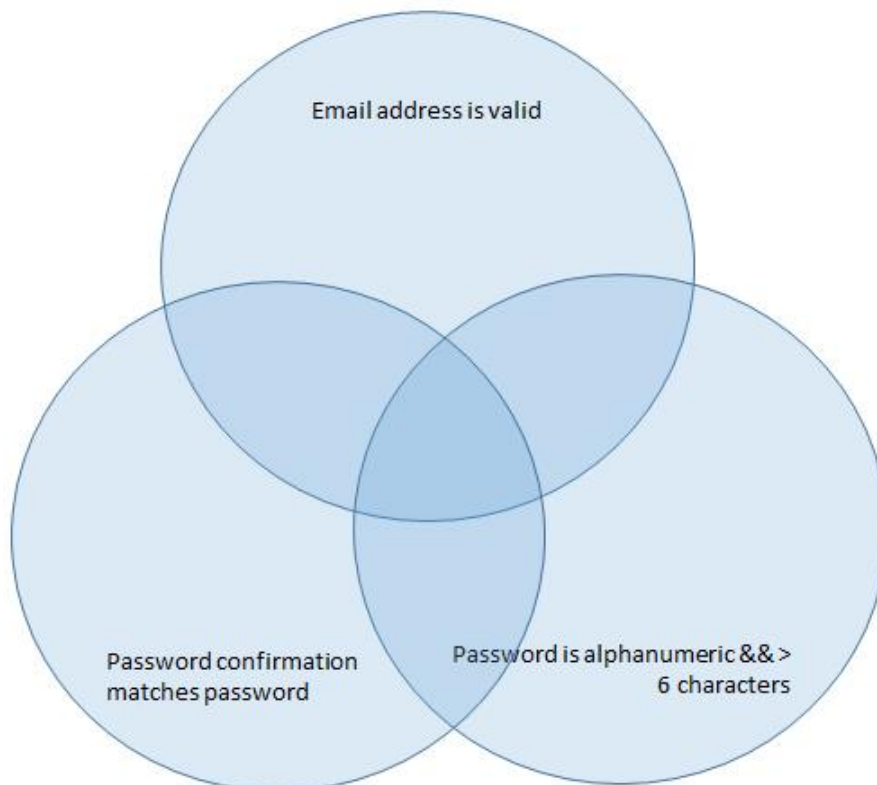
	<u>Shopping basket tests</u>
<input type="radio"/>	Payment accepted
	Payment rejected
	Payment processing timed out
	Sold out during session
<input type="radio"/>	Basket abandoned



codemanship

**TDD Tip #77:** Your software must meaningfully handle any inputs it allows. Any inputs it can't handle should not be allowed.

Email Address	<input type="text" value="jasong@blahblahetc.com"/>
Password	<input type="password" value="*****"/>
Confirm Password	<input type="password" value="*****"/>
<input type="button" value="Register"/>	



codemanship



**TDD Tip #78:** Listen to your tests. Complex set-ups are trying to telling you a class has too many dependencies and probably knows/does too much

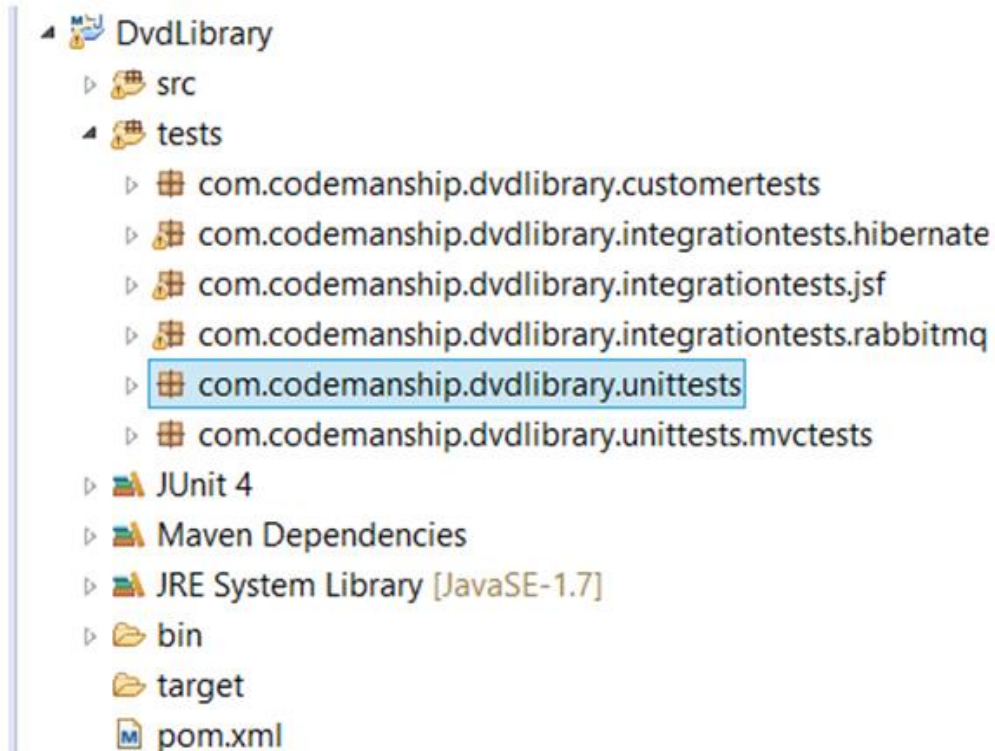
```
@Test
public void checkoutNeedsAWholeBunchOfStuff() {
    PaymentProcessor paymentProcessor = new PayPalProcessor();
    Address address = new Address("1 High Street, London, WC1 1WC");
    Customer customer = new Customer("Jason", "Gorman", address);
    Warehouse warehouse = mock(Warehouse.class);
    Movie movie = new Movie("Star Wars", 10.99);
    when(warehouse.checkStock(movie)).thenReturn(1);
    ReviewService imdb = mock(ReviewService.class);
    when(imdb.getRating(movie)).thenReturn(9.0);
    OrderDAO orderDAO = mock(OrderDAO.class);
    DiscountCalculator discounter = new HolidaySeasonDiscounter();
    Logistics logistics = mock(Logistics.class);
    when(logistics.arrangeShipping(customer)).thenReturn(new ShippingNote(customer));
    Marketing marketing = mock(Marketing.class);
    Logger logging = mock(Logger.class);
    ShoppingBasket basket = new ShoppingBasket(
        paymentProcessor,
        customer,
        warehouse,
        imdb,
        orderDAO,
        discounter,
        logistics,
        marketing,
        logging);

    basket.add(movie, 1);
    basket.checkout();
    Order order = basket.getOrder();
    assertEquals(OrderStatus.PAID, order.getStatus());
}
```



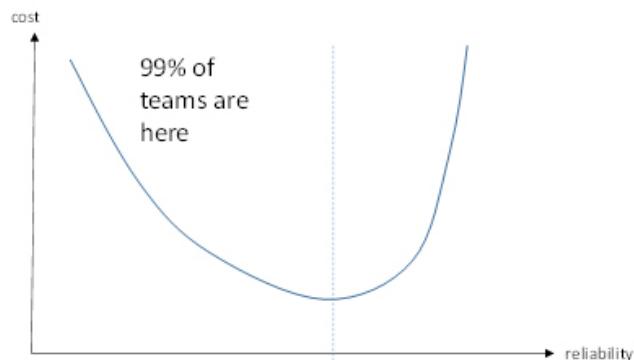
codemanship

## TDD Tip #79: Organise your test suites to make it easy to find and run different kinds of tests

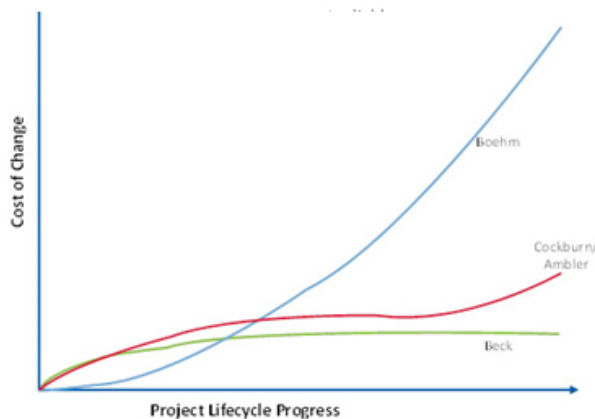


codemanship

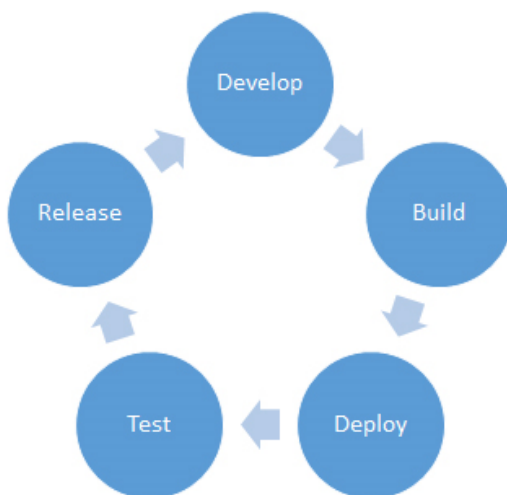
**TDD Tip #80:** TDD isn't what matters. The results you can get with TDD are what matters.



TDD can not only deliver more reliable software, but it can also save you time & money



TDD can help teams sustain the pace of innovation by flattening the cost of change curve

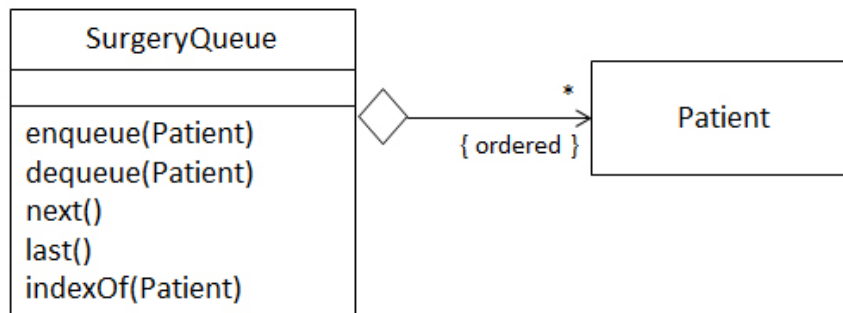


The small cycles of TDD enable Continuous Delivery, so your business can outlearn the competition



codemanship

**TDD Tip #81:** You don't need to write tests for every method of every class. Lead with useful behaviour, and let these details follow



```

public class SurgeryQueueTests {

+   public void enqueuePatient() {..}
+   public void dequeuePatient() {..}
+   public void containsEnqueuedPatient() {..}
+   public void doesntContainPatientIfNotEnqueued() {..}
+   public void mostRecentlyEnqueuedIsLastInQueue() {..}
+   public void leastRecentlyEnqueuedIsNextInQueue() {..}
+   public void findsIndexOfPatientInQueue() {..}
+   public void indexOfPatientNotFoundInQueue() {..}

}
  
```



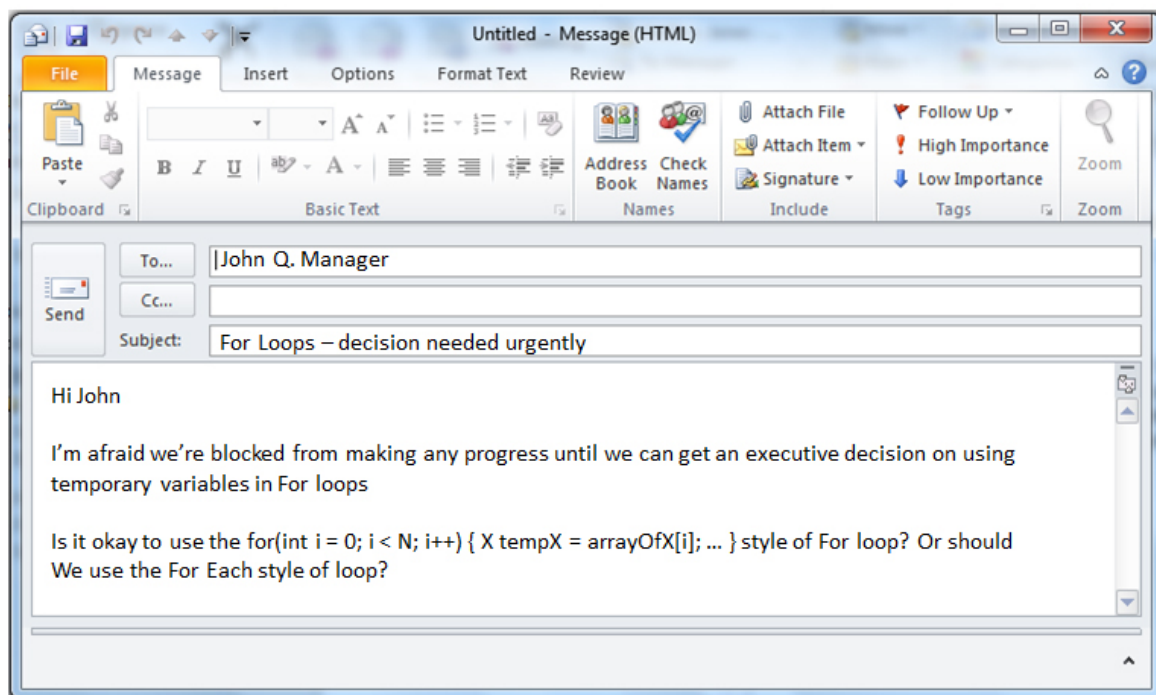
```

5 public class SurgeryQueueTests {
5
3+ public void patientArrivesInWaitingRoom() {..}
L
3+ public void doctorSeesNextPatient() {..}
5
3+ public void patientEnquiresAboutPositionInQueue() {..}
L
3+ public void patientLeavesWaitingRoom() {..}
5
7 }
3
  
```



codemanship

**TDD Tip #82:** TDD is a technical decision. If your boss forbids you to do TDD, escalate every technical decision to them until they insist *\*you\** decide



codemanship



**TDD Tip #83:** Be clear on the difference between a mock, a stub, and a dummy. What matters is how it's used, not how it was created.

```
@Test
public void membersAreEmailedAboutNewDonatedTitle() {
    // this is a STUB because it returns test-specific data
    Title title = mock(Title.class);
    when(title.getName()).thenReturn("The Abyss");

    // this is a MOCK, because we're testing the
    // interaction with it
    EmailQueue emailQueue = mock(EmailQueue.class);

    // this is a DUMMY because it's required to compile
    // the code and run the test, but it's not relevant
    // to the test
    Logger logger = mock(Logger.class);

    Library library = new Library(emailQueue, logger);

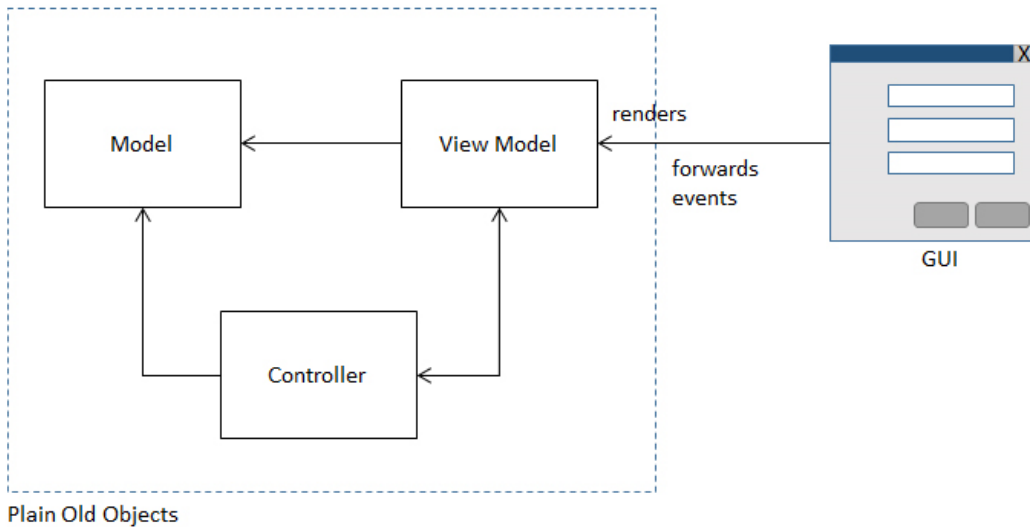
    library.donate(title);

    verify(emailQueue).send("All",
        "A new title has been added you might like",
        "The Abyss");
}
```



codemanship

**TDD Tip #84:** Introducing a “view model” to represent your user interface makes it possible to unit test the logic of the user’s experience



```

public class RaceMeetViewTests {

    @Test
    public void placingBetOnRunner() {
        // build view from domain object data
        List<Horse> runners = new ArrayList<>();
        Horse runner = new Horse("Tea Biscuit");
        runners.add(runner);
        BettingController controller = mock(BettingController.class);
        RaceMeet race = new RaceMeet("Aintree",
                                    "1/7/2017",
                                    "14:00",
                                    runners);
        RaceMeetView view = new RaceMeetView(controller, race);

        // set view state for user input
        view.setSelectedRunner("Tea Biscuit");
        view.setBetAmount(100.00);

        // perform logical user action
        view.bet();

        // test that right system behaviour is invoked
        verify(controller).placeBet(race, runner, 100.00);
    }
}
  
```



codemanship

**TDD Tip #85:** The best way to know if a developer can really do TDD is to watch them do TDD



If you want to know if someone can really juggle, ask to see them juggle



codemanship

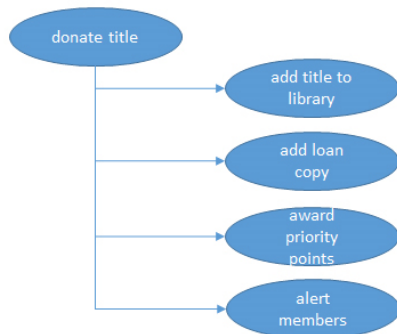
**TDD Tip #86:** The key to good OO design is to start by identifying the work the software needs to do. Objects that do the work come later.

**Given** a DVD title that's already in the library,

**When** a member donates a copy of that title,

**Then...**

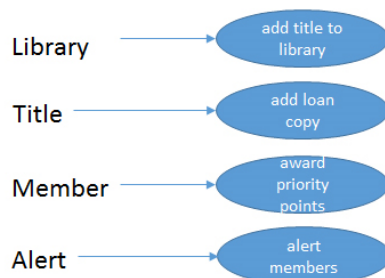
1. The title is added to the library
2. A default rental copy is added to the title
3. The donor is awarded 10 priority points
4. Members who expressed an interest in matching titles are alerted by email



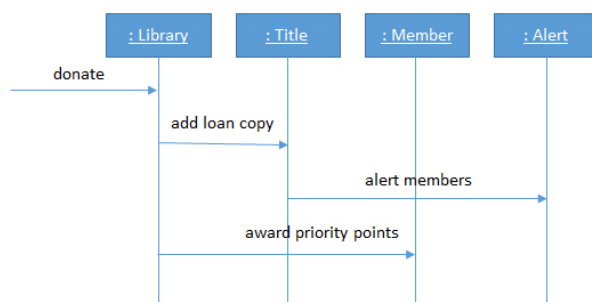
1. Identify work required  
(responsibilities)



2. Identify candidate objects  
that could do the work  
(roles)



3. Assign responsibilities to  
objects that have the  
knowledge required to do  
the work

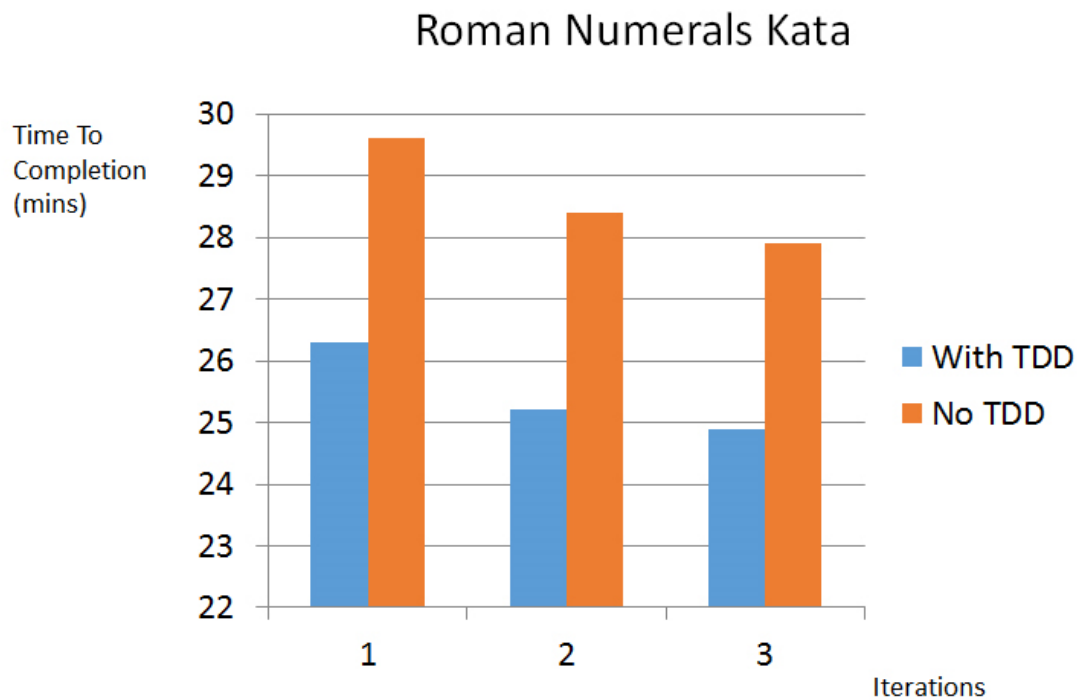


4. Figure out how objects will  
collaborate to coordinate the  
work (collaborations)



codemanship

**TDD Tip #87:** TDD can help us see through the illusion of being 'done' sooner when we cut corners



<http://www.codemanship.co.uk/parlezuml/blog/?postid=1021>



codemanship



**TDD Tip #88:** If your test code is difficult to change, your software is difficult to change. Don't skimp on test code design & refactoring.

```
@Test
public void squareRootOfZeroIsZero() {
    assertEquals(0, Maths.sqroot(0), 0.00001);
}

@Test
public void squareRootOfOneIsOne() {
    assertEquals(1, Maths.sqroot(1), 0.00001);
}

@Test
public void squareRootOfFourIsTwo() {
    assertEquals(2, Maths.sqroot(4), 0.00001);
}

@Test
public void squareRootOfNineIsThree() {
    assertEquals(3, Maths.sqroot(9), 0.00001);
}

@Test
public void squareRootOfSixteenIsFour() {
    assertEquals(4, Maths.sqroot(16), 0.00001);
}

@Test
public void squareRootOfOneQuarterIsOneHalf() {
    assertEquals(0.5, Maths.sqroot(0.25), 0.00001);
}
```

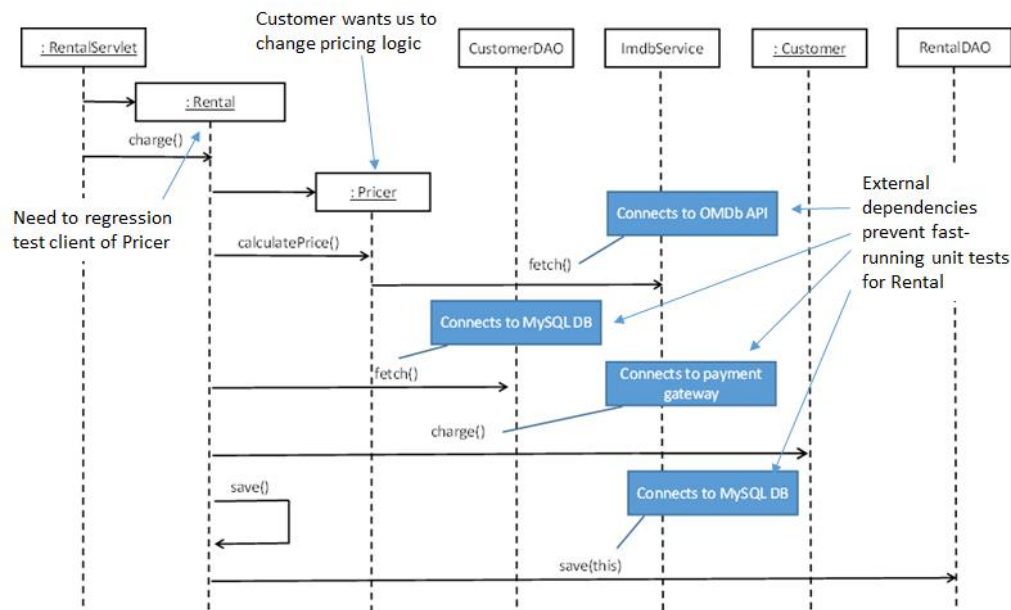
refactored

```
@Test
@Parameters({"0", "1", "4", "9", "16", "0.25"})
public void squareOfSquareRootIsSameAsInput(double input) {
    double sqrootSquared = Maths.sqroot(input)
        * Maths.sqroot(input);
    assertEquals(input, sqrootSquared, 0.00001);
}
```



codemanship

**TDD Tip #89:** If asked to change legacy code (code that has no automated tests), priority one is to get tests around anything that change might break



Write automated integration tests for Rental::charge()

Make external dependencies *swappable* using dependency injection, so we can turn those slow integration tests into fast-running unit tests

```

public class Rental {

    private float amountCharged;
    private final DAO customerDAO;
    private final Pricer pricer;
    private final DAO rentalDAO;

    public Rental(Pricer pricer, DAO customerDAO, DAO rentalDAO) {
        this.customerDAO = customerDAO;
        this.pricer = pricer;
        this.rentalDAO = rentalDAO;
    }

    public void charge() {
        Customer customer = customerDAO.fetch();
        amountCharged = pricer.calculatePrice();
        customer.charge(amountCharged);
        save();
    }

    private void save() {
        rentalDAO.save(this);
    }
}
  
```



codemanship

**TDD Tip #90:** 9/10 developers who claim they can do TDD give themselves away with some of these classic tell-tale signs of inexperience

- Plasters CV with meaningless references to TDD
- Configures project for stuff *\*might\** need later
- Talks about “testing”, not design
- **Starts writing solution code first**
- Doesn't separate test and solution code
- Writes multiple failing tests at a time
- Makes the test fail with *fail()*
- Doesn't check that the test fails first
- Starts with a complicated example
- **Writes a general solution for a single test**
- **Doesn't refactor when the code calls for it**
- Doesn't refactor test code
- Writes “design-driven tests”
- Writes test code that doesn't clearly communicate intent
- Doesn't know commonly used shortcuts in chosen IDE
- Doesn't use available automated refactorings
- Doesn't know the difference between a stub, a mock and a dummy
- Doesn't use test doubles & dependency injection for external dependencies
- Tests mocks and stubs



codemanship

**TDD Tip #91:** Do not conclude from your initial inability to make TDD work for you that TDD does not work

After 3 frustrating lessons, Phil concluded that the tuba is not a viable musical instrument



codemanship



**TDD Tip #92:** Commenting out failing tests does not make the problem go away



codemanship



**TDD Tip #93:** Don't get bogged down in unit tests. Take a step back to see the bigger picture.



codemanship


**TDD Tip #94:** Don't waste time trying to force-feed TDD to colleagues. Let them *see* the benefits and make up their own minds.



codemanship

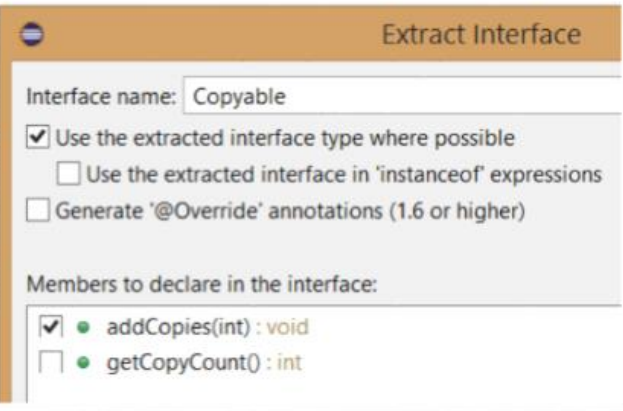
**TDD Tip #95:** In a test-driven approach, we declare source code for 2 reasons: to fix a broken test, or when we're refactoring

```
@Test
public void donatedTitlesAddedToLibrary()
{
    Library library;
    assertEquals(0, library.getTitles().size());
}
```

A suggestion menu from IntelliJ IDEA is shown, listing options: 'Create class 'Library'', 'Create interface 'Library'', 'Create enum 'Library'', and 'Add two parameter 'Liked' to 'VideoLike'.

We declare the class `Library` so our test code will compile

```
2 public class Title {
3
4     private int copies;
5
6 }
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

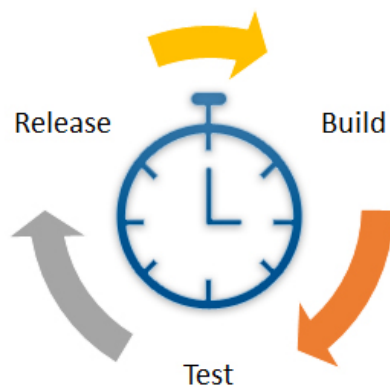
The 'Extract Interface' dialog is shown. The 'Interface name' is 'Copyable'. The 'Use the extracted interface type where possible' checkbox is checked. The 'Members to declare in the interface' section shows 'addCopies(int) : void' and 'getCopyCount() : int' both checked.

We extract the interface *Copyable* to present a client-specific interface to `Library`

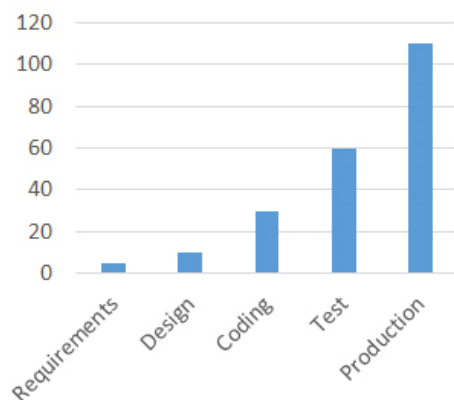


codemanship

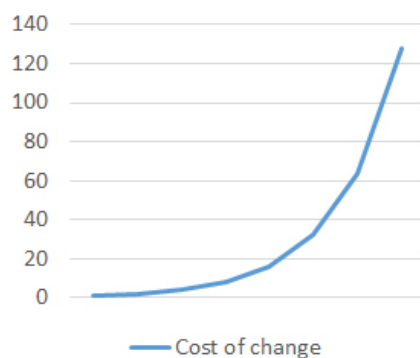
**TDD Tip #96:** The most important TDD metrics are not about its effect on code, but its effect on the business



Building software in micro-iterations, always ending with something that's potentially shippable means we can dramatically reduce **delivery cycle & lead times**



TDD can produce more reliable software at minimal – often no - extra cost. Lower **bug counts** in releases mean happier users, and more **time for new feature development** in subsequent releases

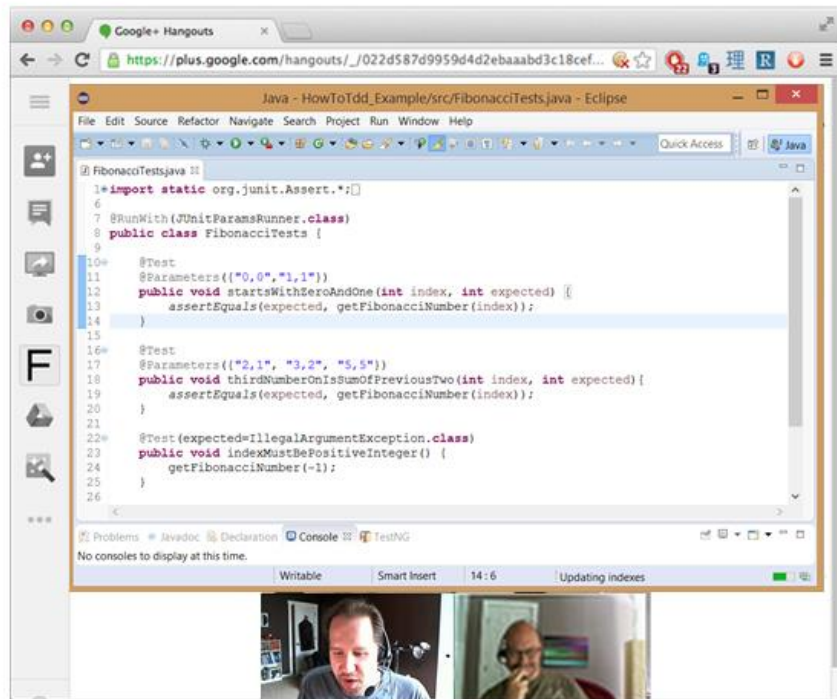


The automated tests TDD creates, together with its tendency to produce simpler code, can dramatically flatten the **cost of change** curve for the software, meaning we can sustain the pace of innovation for longer, and effectively outlearn the competition



codemanship

**TDD Tip #97:** Good TDD-ers are still hard to find. Save everyone's time by starting candidates with a 10-minute remote pairing session



codemanship



**TDD Tip #98:** TDD starts before we even write a test. Use examples to test your understanding of the customer's requirements.



codemanship

**TDD Tip #99:** In a test-driven approach, design patterns are discovered as a product of refactoring rather than planned with up-front design

```

@Test
@Parameters
public void writesCorrectKindOfResponse(ResponseKind responseKind,
                                         String startsWithString) {
    Customer customer = new Customer("Kent Beck");
    String response = new ResponseWriter().createResponse(customer, responseKind);
    assertTrue(response.startsWith(startsWithString));
}

private Object[] parametersForWritesCorrectKindOfResponse(){
    return new Object[][]{
        {ResponseKind.HTML, "<html>"},
        {ResponseKind.XML, "<customer>"},
        {ResponseKind.STRING, "Customer"}
    };
}

public class ResponseWriter {

    public String createResponse(Customer customer, ResponseKind responseKind) {
        String response = "";
        switch(responseKind){
            case HTML:
                response = new HtmlSerializer().serializeToHtml(customer);
                break;
            case XML:
                response = new XmlSerializer().serializeToXml(customer);
                break;
            default:
                response = new StringSerializer().serializeToString(customer);
        }
        return response;
    }
}

```



Replace type code with  
Strategy pattern

```

@Test
@Parameters
public void writesCorrectKindOfResponse(Serializer serializer,
                                         String startsWithString) {
    Customer customer = new Customer("Kent Beck");
    String response = new ResponseWriter().createResponse(customer, serializer);
    assertTrue(response.startsWith(startsWithString));
}

private Object[] parametersForWritesCorrectKindOfResponse(){
    return new Object[][]{
        {new HtmlSerializer(), "<html>"},
        {new XmlSerializer(), "<customer>"},
        {new StringSerializer(), "Customer"}
    };
}

public class ResponseWriter {

    public String createResponse(Customer customer, Serializer serializer) {
        return serializer.serialize(customer);
    }
}

```



codemanship

**TDD Tip #100:** If changing the contents of a file can break your software, you should consider test-driving it

```
<?xml version="1.0"?>
<!DOCTYPE project>
<project name="Boilerplate Build" default="build" basedir=".."><!-- one back since we're in build/ -->

    <!-- load shell environment -->
    <property environment="ENV" />

    <!-- load property files -->
    <property file="build/config/project.properties"/><property file="build/config/default.properties"/>

    <!-- Load in Ant-Contrib to give us -->
    <!-- the .jar file is located in the -->
    <taskdef resource="net/sf/antcontrib" classpath=
        <classpath>
            <pathelement location="${basedir}/lib/ant-contrib.jar"/>
        </classpath>
    </taskdef>

    <!-- merge the stylesheet properties -->
    <var name="stylesheet-files" value="build/config/stylesheet-files.xml"/>

    <!-- Load in Hibernate Configuration DTD 3.0 -->
    <!-- http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd -->
    <!-- Use Thread local contextual sessions -->
    <!-- thread -->
    </property>
```



codemanship

## **TDD Tip #101:** TDD isn't compulsory. The choice is yours.



Choose to minimise costly misunderstandings about requirements



Choose to deliver more reliable and more maintainable code



Choose to have code that's always shippable, for faster feedback cycles and shorter lead times



Choose to be able to sustain the pace of innovation on your product for longer & outlearn the competition



**codemanship**

For the most practical, hands-on TDD training, visit [www.codemanship.com](http://www.codemanship.com)