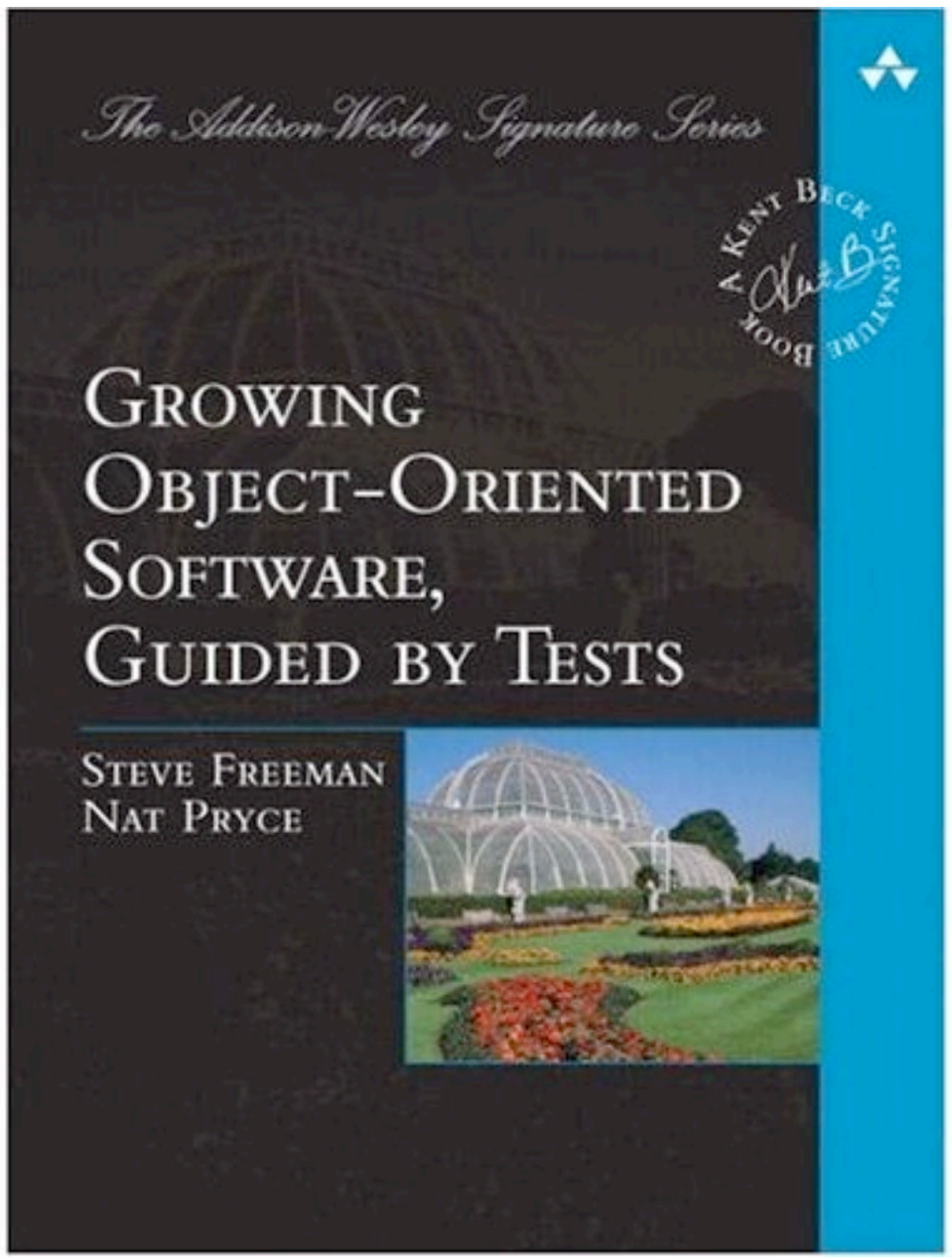
Sustainable Test-Driven Development

Steve Freeman and Nat Pryce

M3P

steve.freeman@m3p.co.uk

Out Now


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce

Why Sustainability Matters (a)

```
public class ExchangeRateUploaderTest extends EasyMockTestCase {
    private Logger logger;
    private CurrencyManager mockCurrencyManager;
    private ExchangeRateManager mockExchangeRateManager;
    private PriceManagerFactory mockPriceManagerFactory;
    private PriceManager mockPriceManager;
    private GodObject mockGod;
    private DatabaseFacade mockPersistenceManager;
    private DatabaseFacade mockFrameworkPersistenceManager;
    private CyclicProcessManager mockCyclicProcessManager;
    private SystemVariableManager mockSystemVariableManager;
    private ScreenManager mockScreenManager;
    private Registry registry;
    private User adminUser;
    private Server server;

    private ExchangeRateUploader newExchangeRateUploader(CyclicProcessThread thread) {
        return new ExchangeRateUploader(thread) {
            @Override protected void initializeAction() throws FrameworkException {
                // Does nothing to prevent excessive mocking
            }
            @Override public Logger getLogger() { return logger; }
            @Override protected void setLogMDC() { }
            @Override protected User getUser() { return adminUser; }
            @Override protected CurrencyManager newCurrencyManager() { return mockCurrencyManager; }
            @Override protected PriceManagerFactory newPriceManagerFactory() { return mockPriceManagerFactory; }
            @Override protected CyclicProcessManager newCyclicProcessManager() { return mockCyclicProcessManager; }
            @Override protected DatabaseFacade newPersistenceManager() { return mockPersistenceManager; }
            @Override protected Registry newTaskPerformanceRegistry() { return registry; }
            @Override public PriceDataManager getPriceDataManager() { return null; }
            @Override protected ExchangeRateManager newExchangeRateManager() { return mockExchangeRateManager; }
        };

    }
```

steve.freeman@m3p.co.uk

Why Sustainability Matters (b)

```
public void testInternalAction() throws FrameworkException {
    mockGod = addMock(GodObject.class);
    expect(mockGod.getPriceDataManager()).andStubReturn(null);
    mockPersistenceManager = addMock(DatabaseFacade.class);
    registry = addMock(Registry.class);
    adminUser = new TestUser("Admin", "", "", new TestCompany("company"), "");
    mockCyclicProcessManager();

    registry.finalizeThisThread(isA(String.class), isA(String.class));
    Date now = DateUtils.trimMinutesAndSecondsFromDate(new Date());

    mockSystemVariableManager();
    mockLogger();
    mockContextPersistenceManager();
    mockPriceManager();

    CyclicProcessThread thread = mockUserStateAndGetCyclicProcessThread();

    String primeName = "prime";
    String aName = "a";
    String otherName = "other";

    Currency primeCurrency = new TestCurrency(primeName, true);
    Currency aCurrency = new TestCurrency(aName, true);
    Currency otherCurrency = new TestCurrency(otherName, false);

    FXCurrencyPair aCurrencyPair = new FXCurrencyPair(primeCurrency, aCurrency);
    FXCurrencyPair otherCurrencyPair = new FXCurrencyPair(otherCurrency, primeCurrency);

    setupCurrencyManager(primeCurrency, aCurrency, otherCurrency);
    mockExchangeRateManager = addMock(ExchangeRateManager.class);

    mockGetFXRatesAtDatesForCurrencies(now, aCurrencyPair, otherCurrencyPair);

    FrameworkNumber aCurrencyValue = new FrameworkNumber("5");
    FrameworkNumber otherCurrencyValue = new FrameworkNumber("2");
    ExchangeRate aCurrencyRate = new ExchangeRate(primeCurrency, aCurrency, aCurrencyValue, now);
    ExchangeRate otherCurrencyRate = new ExchangeRate(otherCurrency, primeCurrency, otherCurrencyValue, now);
    expect(mockCurrencyManager.getParentToFractionalCurrencyMapForFractionalCurrency()).andStubReturn(newMap());

    expect(
        mockExchangeRateManager.saveExchangeRate(null, new FrameworkString(primeName), new FrameworkString(aName), aCurrencyValue,
            new FrameworkDate(now))).andReturn(null);
    expect(
        mockExchangeRateManager.saveExchangeRate(null, new FrameworkString(otherName), new FrameworkString(primeName),
            otherCurrencyValue, new FrameworkDate(now))).andReturn(null);

    Map<String, ExchangeRate> out = new HashMap<String, ExchangeRate>();
    out.put("primea", aCurrencyRate);
    out.put("otherprime", otherCurrencyRate);
    expect(mockPriceManager.getLatestExchangeRates(newList(aCurrencyPair, otherCurrencyPair))).andReturn(out);

    mockPMFactoryCleanup();

    replayMocks();

    ExchangeRateUploader uploader = newExchangeRateUploader(thread);
    uploader.initialise();
    uploader.run();
}
```

steve.freeman@m3p.co.uk ©2009

Why Sustainability Matters (c)

```
private void mockPMFactoryCleanup() {
    PersistenceFactory mockPersistenceFactory = addMock(PersistenceFactory.class);
    mockPersistenceFactory.purgeAllStateForThisThread();
    expect(mockGod.getPersistenceFactory()).andReturn(mockPersistenceFactory).anyTimes();
    expect(mockPersistenceFactory.getExceptionsInRequest()).andReturn(Collections.<Throwable>emptyList()).times(1);
}

private void mockCyclicProcessManager() throws CyclicProcessException {
    mockCyclicProcessManager = addMock(CyclicProcessManager.class);
    expect(mockGod.getCyclicProcessManager()).andStubReturn(mockCyclicProcessManager);
    mockCyclicProcessManager.updateServerCyclicProcessCurrentRunStatus(isA(String.class),
        isA(LISTENER_STATUS.class), isA(String.class), isA(Double.class), isA(Date.class));
    expectLastCall().anyTimes();
    server = addMock(Server.class);
    expect(mockCyclicProcessManager.getThisServer()).andStubReturn(server);
}

private void setupCurrencyManager(Currency primeCurrency, Currency aCurrency, Currency otherCurrency) {
    mockCurrencyManager = addMock(CurrencyManager.class);
    List<Currency> allCurrencies = new ArrayList<Currency>();
    allCurrencies.add(aCurrency);
    allCurrencies.add(primeCurrency);
    allCurrencies.add(otherCurrency);
    expect(mockCurrencyManager.getPrimeCurrency()).andReturn(primeCurrency).anyTimes();
    expect(mockCurrencyManager.getAllParentCurrencies()).andReturn(allCurrencies).times(2);
}

private void mockGetFXRatesAtDatesForCurrencies(Date now, FXCurrencyPair aCurrencyPair,
    FXCurrencyPair otherCurrencyPair) throws CurrencyException
{
    FrameworkNumber originalACurrencyRate = new FrameworkNumber("1.23");
    Map<FXCurrencyPair, Collection<Date>> currencyPairAndDatesMap = new HashMap<FXCurrencyPair, Collection<Date>>();
    currencyPairAndDatesMap.put(aCurrencyPair, Arrays.asList(now));
    currencyPairAndDatesMap.put(otherCurrencyPair, Arrays.asList(now));
    FXCurrencyPairRates outputObj = addMock(FXCurrencyPairRates.class);
    expect(outputObj.rateMapSize()).andReturn(5).anyTimes();
    expect(outputObj.getActualPriceDateForCurrencyPair(aCurrencyPair, now)).andReturn(null).once();
    expect(outputObj.getRateFromFxRateMap(now, aCurrencyPair)).andReturn(originalACurrencyRate).once();
    expect(outputObj.getActualPriceDateForCurrencyPair(otherCurrencyPair, now)).andReturn(null).once();
    expect(outputObj.getRateFromFxRateMap(now, otherCurrencyPair)).andReturn(originalACurrencyRate);
    expect(mockExchangeRateManager.getFXRatesAtDatesForCurrencies(currencyPairAndDatesMap)).andReturn(outputObj);
}
```

steve.freeman@m3p.co.uk

Why Sustainability Matters (d)

```
private CyclicProcessThread mockUserStateAndGetCyclicProcessThread() {
    Role mockAdminRole = addMock(Role.class);
    CyclicProcessThread thread = addMock(CyclicProcessThread.class);
    expect(thread.getAdminRole()).andReturn(mockAdminRole).anyTimes();
    expect(thread.getAdminUser()).andReturn(adminUser).anyTimes();
    thread.interrupt();
    expectLastCall();
    mockScreenManager = addMock(ScreenManager.class);
    expect(mockGod.getScreenManager()).andReturn(mockScreenManager).anyTimes();
    mockScreenManager.setThreadSignedOnState(new SignedOnState(adminUser, mockAdminRole, false));
    expectLastCall().anyTimes();
    expect(thread.getGod()).andReturn(mockGod).anyTimes();
    expect(thread.getShutdownInProgress()).andReturn(false).anyTimes();
    return thread;
}

private void mockContextPersistenceManager() {
    mockFrameworkPersistenceManager = addMock(DatabaseFacade.class);
    expect(mockGod.getDatabaseFacade()).andReturn(mockFrameworkPersistenceManager).anyTimes();
    mockFrameworkPersistenceManager.beginNewSession();
    expectLastCall().anyTimes();
}

private void mockPriceManager() throws PriceException {
    mockPriceManagerFactory = addMock(PriceManagerFactory.class);
    mockPriceManager = addMock(PriceManager.class);
    expect(mockPriceManagerFactory.newPriceManager(mockFrameworkPersistenceManager,
                                                   mockSystemVariableManager, null))
        .andReturn(mockPriceManager).once();
}

private void mockSystemVariableManager() {
    mockSystemVariableManager = addMock(SystemVariableManager.class);
    expect(mockGod.getSystemVariableManager()).andReturn(mockSystemVariableManager).anyTimes();
    expect(mockSystemVariableManager.getSystemVariable(CYCLIC_PROCESS_LISTENER_HEART_BEAT_TOLERANCE, "30000"))
        .andReturn("30000").anyTimes();
}

private void mockLogger() {
    logger = addMock(Logger.class);
    logger.info(isA(String.class)); expectLastCall().atLeastOnce();
    logger.debug(isA(String.class)); expectLastCall().atLeastOnce();
}
}
```

steve.freeman@m3p.co.uk ©2009

Test Readability

To design is to communicate clearly by whatever means you can control or master.

—Milton Glaser

Test Names Describe Features

```
public class TargetObjectTest {
  @Test public void isReady() {
  @Test public void choose() {
  @Test public void choose1() {
```

```
public class TargetObject {
  public void isReady() {
  public void choose(Picker picker) {
```

M3P

steve.freeman@m3p.co.uk ©2009

Test Names Describe Features

```
public class ListTests {
  @Test public void
  holdsItemsInTheOrderTheyWereAdded() {
  @Test public void
  canHoldMultipleReferencesToTheSameItem() {
  @Test public void
  throwsAnExceptionWhenRemovingAnItemItDoesntHold() {
```

steve.freeman@m3p.co.uk

Canonical Test Structure

```
public class StringTemplateTest {
 @Test public void expandsMacrosSurroundedWithBraces() {
    StringTemplate template = new StringTemplate("{a}{b}"); Setup
    HashMap<String,Object> macros = new HashMap<String,Object>();
    macros.put("a", "A"); macros.put("b", "B");

    String expanded = template.expand(macros);          Execute

    assertThat(expanded, equalTo("AB"));                Assert
                                                        Teardown
  }
}
```

M3P

steve.freeman@m3p.co.uk ©2009

Streamline the Test Code

```
assertThat(instruments,
           hasItem(instrumentWithPrice(greaterThan(81))));
```

steve.freeman@m3p.co.uk ©2009

Narrow Assertions and Expectations

```
oneOf(failureReporter).cannotTranslateMessage(
                          with(SNIPER_ID), with(badMessage),
                          with(any(RuntimeException.class)));
```

steve.freeman@m3p.co.uk

Self-Describing Variables

```
final static Chat UNUSED_CHAT = null;
```

```
final static int INVALID_ID = 666;
```

steve.freeman@m3p.co.uk ©2009

Constructing Complex Test Data

Many attempts to communicate are nullified by saying too much.

—Robert Greenleaf

The Problem With Object Structures

```
Order order = new Order(
  new Customer("Sherlock Holmes",
    new Address("221b Baker Street",
                "London",
                new PostCode("NW1", "3RX"))));
order.addLine(new OrderLine("Deerstalker Hat", 1));
order.addLine(new OrderLine("Tweed Cape", 1));
```

```
Order order1 = ExampleOrders.newDeerstalkerAndCapeAndSwordstickOrder();
Order order2 = ExampleOrders.newDeerstalkerAndBootsOrder();
```

steve.freeman@m3p.co.uk ©2009

Test Data Builder: Add Indirection

```
public class OrderBuilder {
  private Customer customer = new CustomerBuilder().build();
  private List<OrderLine> lines = new ArrayList<OrderLine>();
  private BigDecimal discountRate = BigDecimal.ZERO;

  public OrderBuilder withCustomer(Customer customer) {
    this.customer = customer;
    return this;
  }
  public OrderBuilder withOrderLines(OrderLines lines) {
    this.lines = lines;
    return this;
  }
  public OrderBuilder withDiscount(BigDecimal discountRate) {
    this.discountRate = discountRate;
    return this;
  }
  public Order build() {
    Order order = new Order(customer);
    for (OrderLine line : lines) order.addLine(line);
    order.setDiscountRate(discountRate);
    return order;
  }
}
```

M3P

steve.freeman@m3p.co.uk

Only Need To Include Relevant Values

```
new OrderBuilder()
  .fromCustomer(
      new CustomerBuilder()
       .withAddress(new AddressBuilder().withNoPostcode().build())
       .build())
  .build();
```

steve.freeman@m3p.co.uk ©2009

Named Methods Make Mistakes Obvious

```
new AddressBuilder()
    .withStreet("221b Baker Street")
    .withStreet2("London")
    .withPostCode("NW1 6XE")
    .build();
```

M3P

steve.freeman@m3p.co.uk

Use Builders to Create Similar Objects

```
OrderBuilder hatAndCape = new OrderBuilder()
  .withLine("Deerstalker Hat", 1)
  .withLine("Tweed Cape", 1);
```

```
Order orderWithDiscount    = hatAndCape
                               .but().withDiscount(0.10).build();
Order orderWithGiftVoucher = hatAndCape
                               .but().withGiftVoucher("abc").build();
```

M3P

steve.freeman@m3p.co.uk

Compacting Construction

```
Order order = anOrder()
                .from(aCustomer()
                        .with(anAddress().withNoPostcode()))
                .build();
```

```
Address aLongerAddress = anAddress()
                           .withStreet("221b Baker Street")
                           .withCity("London")
                           .with(postCode("NW1", "3RX"))
                           .build();
```

steve.freeman@m3p.co.uk ©2009

Refactor To Builders

```
@Test public void reportsTotalSalesOfOrderedProducts() {
  sendAndProcess(anOrder()
                   .withLine("Deerstalker Hat", 1)
                   .withLine("Tweed Cape", 1));
  sendAndProcess(anOrder().withLine("Deerstalker Hat", 1));

  TotalSalesReport report = gui.openSalesReport();
  report.checkDisplayedTotalSalesFor("Deerstalker Hat", is(equalTo(2)));
  report.checkDisplayedTotalSalesFor("Tweed Cape", is(equalTo(1)));

void sendAndProcess(OrderBuilder orderDetails) {
  Order order = orderDetails
                  .withDefaultCustomersReference(nextCustomerReference())
                  .build();
  requestSender.send(order);
  progressMonitor.waitForCompletion(order);
}
```

M3P

steve.freeman@m3p.co.uk

What, Not How

```
@Test public void reportsTotalSalesOfOrderedProducts() {
  havingReceived(anOrder()
                   .withLine("Deerstalker Hat", 1)
                   .withLine("Tweed Cape", 1));
  havingReceived(anOrder().withLine("Deerstalker Hat", 1));

  TotalSalesReport report = gui.openSalesReport();
  report.displaysTotalSalesFor("Deerstalker Hat", equalTo(2));
  report.displaysTotalSalesFor("Tweed Cape", equalTo(1));
}
```

steve.freeman@m3p.co.uk ©2009

Test Diagnostics

Mistakes are the portals of discovery.

—James Joyce

Explain Yourself

```
assertEquals(16301, customer.getBalance());
```

ComparisonFailure: expected:<[16301]> but was:<[16103]>

```
assertEquals("balance", 16301, customer.getBalance());
```

*ComparisonFailure: **balance** expected:<[16301]> but was:<[16103]>*

steve.freeman@m3p.co.uk ©2009

Describe Yourself

```
ComparisonFailure: expected:<[a customer account id]>
                   but was:<[id not set]>
```

```
java.lang.AssertionError: payment date
Expected: <Thu Jan 01 01:00:01 GMT 1970>
     got: <Thu Jan 01 01:00:02 GMT 1970>
```

```
Date startDate = namedDate(1000, "startDate");
Date endDate = namedDate(2000, "endDate");

Date namedDate(long timeValue, final String name) {
  return new Date(timeValue) {
   public String toString() { return name; }
  };
}
```

```
java.lang.AssertionError: payment date
Expected: <startDate>
     got: <endDate>
```

steve.freeman@m3p.co.uk ©2009

Tracer Objects

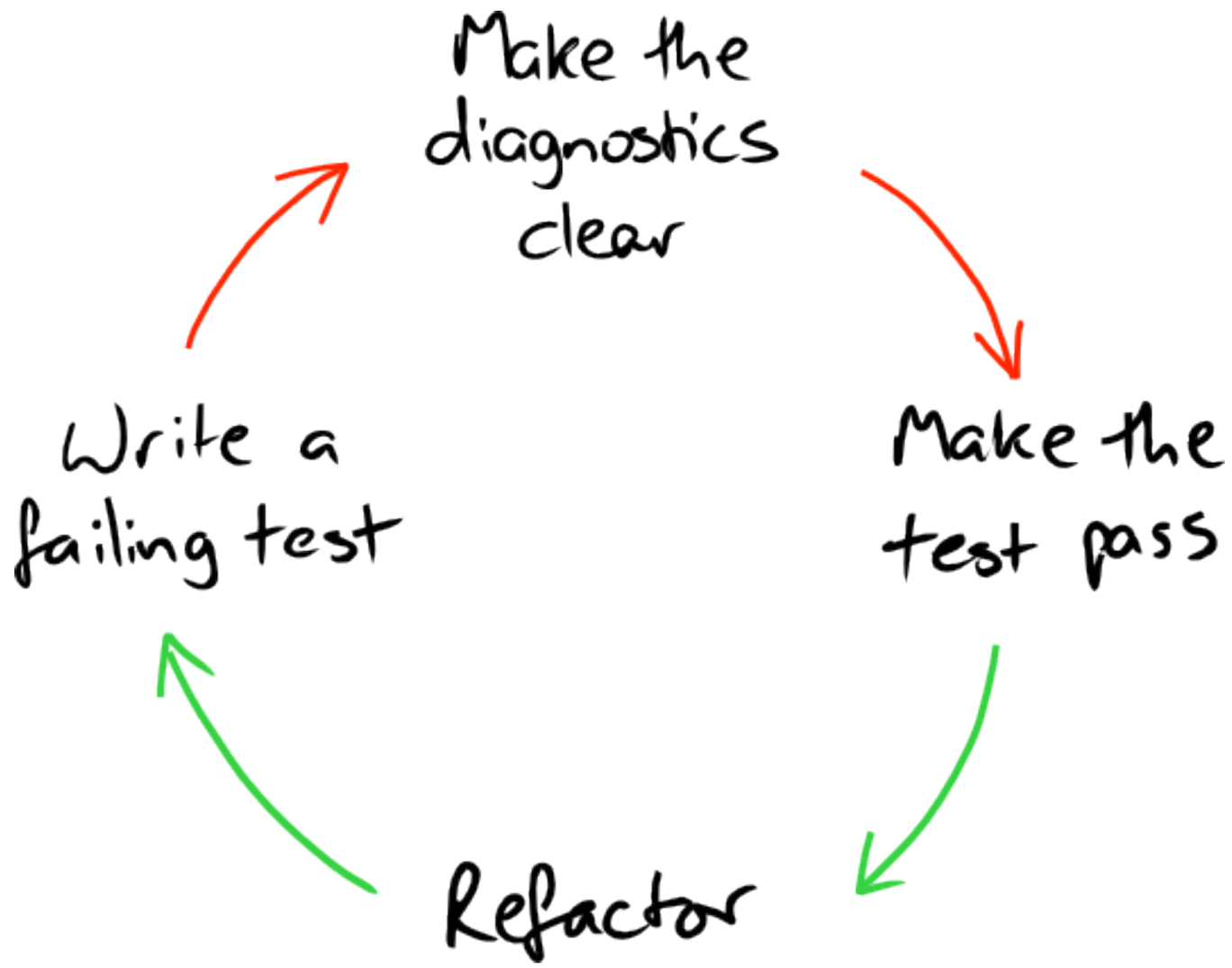
```
@RunWith(JMock.class) public class CustomerTest {
  final LineItem item1 = context.mock(LineItem.class, "item1");
  final LineItem item2 = context.mock(LineItem.class, "item2");
  final Billing billing = context.mock(Billing.class);

  @Test public void requestsInvoiceForPurchasedItems() {
    context.checking(new Expectations() {{
      oneOf(billing).add(item1);
      oneOf(billing).add(item2);
    }});
    customer.purchase(item1, item2);
    customer.requestInvoice(billing);
  }
}
```

```
not all expectations were satisfied
expectations:
  expected once, already invoked 1 time: billing.add(<item1>)
  ! expected once, never invoked: billing.add(<item2>)
what happened before this:
  billing.add(<item1>)
```

steve.freeman@m3p.co.uk ©2009

Diagnostics Are a First-Class Feature



M3P

steve.freeman@m3p.co.uk

Test Flexibility

Living plants are flexible and tender;
the dead are brittle and dry.
[...]
The rigid and stiff will be broken.
The soft and yielding will overcome.

—Lao Tzu (c.604—531 B.C.)

Specify Precisely What Should Happen and No More

M3P

steve.freeman@m3p.co.uk

Information, Not Representation

```
public interface CustomerBase {
  // Returns null if no customer found
  Customer findCustomerWithEmailAddress(String emailAddress);
```

```
allowing(customerBase).findCustomerWithEmailAddress(theAddress);
                      will(returnValue(null));
```

```
public static final Customer NO_CUSTOMER_FOUND = null;
```

```
public interface CustomerBase {
  Maybe<Customer> findCustomerWithEmailAddress(String emailAddress);
}
```

steve.freeman@m3p.co.uk

Precise Assertions

```
assertThat(“strike price”,
           92, equalTo(instrument.getStrikePrice()));
```

```
  assertThat(“transaction id”,
             instrument.getTransactionId(),
             largerThan(PREVIOUS_TRANSACTION_ID));
```

```
assertThat(failureMessage,
           allOf(containsString("strikePrice=92"),
                 containsString("id=FGD.430"),
                 containsString("is expired")));
```

M3P

steve.freeman@m3p.co.uk ©2009

Precise Expectations

```
oneOf(auction).addAuctionEventListener(with(sniperForItem(itemId)));
```

```
oneOf(auditTrail).recordFailure(with(
                                    allOf(containsString("strikePrice=92"),
                                          containsString("id=FGD.430"),
                                          containsString("is expired"))));
```

steve.freeman@m3p.co.uk ©2009

Allow Queries
Expect Commands

```
ignoring(auditTrail);
allowing(catalog).getPriceForItem(item); will(returnValue(74));

exactly(2).of(order).addItem(item, 74);
```

steve.freeman@m3p.co.uk ©2009

Only Enforce Order When It Matters

```
@Test public void announcesMatchForOneAuction() {
  final AuctionSearcher auctionSearch =
                  new AuctionSearcher(searchListener, asList(STUB_AUCTION1));

  context.checking(new Expectations() {{
    Sequence events = context.sequence("events");
    oneOf(searchListener).searchMatched(STUB_AUCTION1); inSequence(events);
    oneOf(searchListener).searchFinished();             inSequence(events);
   }});

  auctionSearch.searchFor(KEYWORDS);
}
```

steve.freeman@m3p.co.uk ©2009

“Guinea Pig” Objects

```
public class XmlMarshallerTest {
  public static class MarshalledObject {
   private String privateField = "private";
   public final String publicFinalField = "public final";
   // constructors, accessors for private field, etc.
  }

  public static class WithTransient extends MarshalledObject {
    public transient String transientField = "transient";
  }

  @Test public void marshallsAndUnmarshallsSerialisableFields() {
    XMLMarshaller marshaller = new XmlMarshaller();
    WithTransient original = new WithTransient();
    String xml = marshaller.marshall(original);
    AuctionClosedEvent unmarshalled = marshaller.unmarshall(xml);
    assertThat(unmarshalled,
               hasSameSerialisableFieldsAs(original));
  }
}
```

steve.freeman@m3p.co.uk ©2009

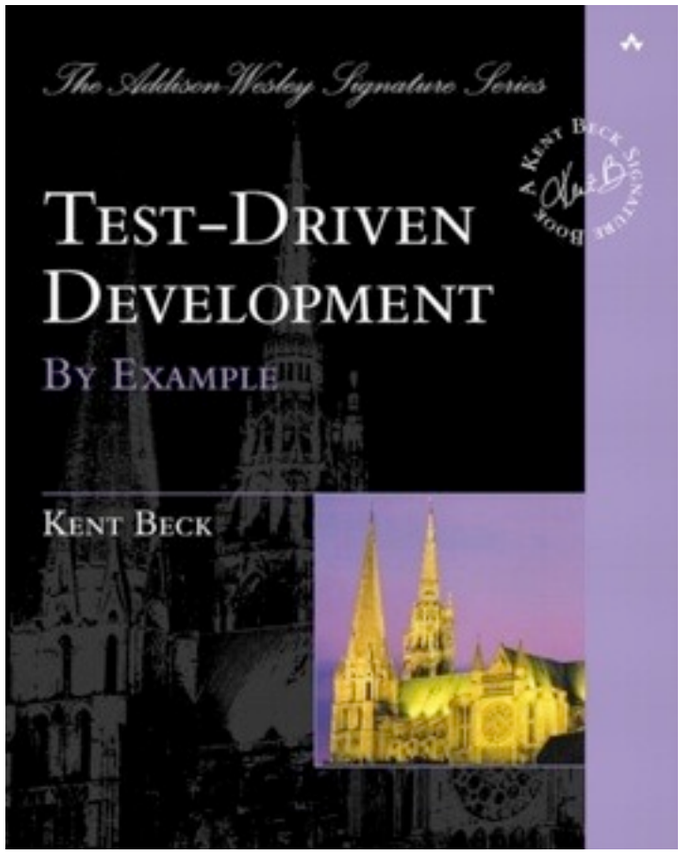
Tests Are Code Too

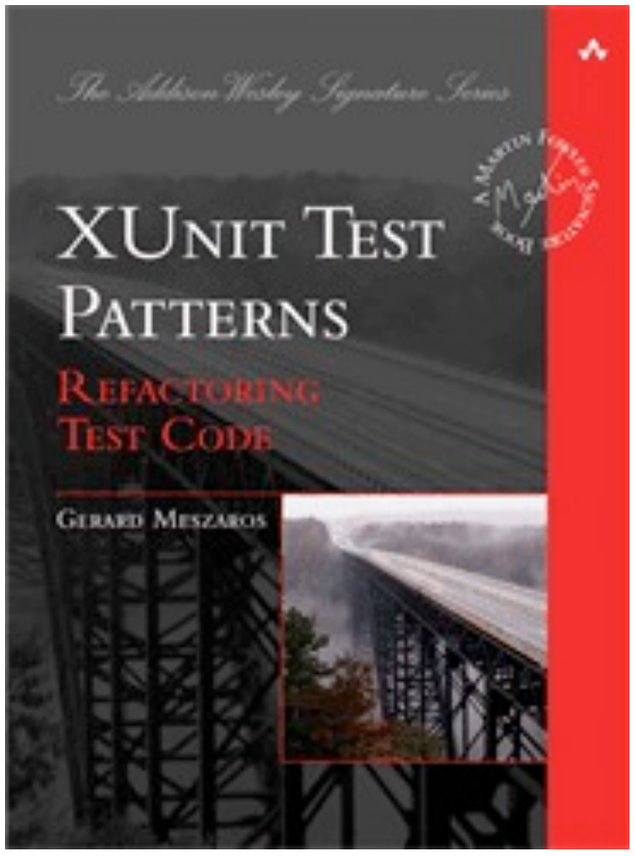
- Expressiveness over convenience
- Refactor and abstract
- Focus on what matters
- If it’s hard to test, that’s a clue

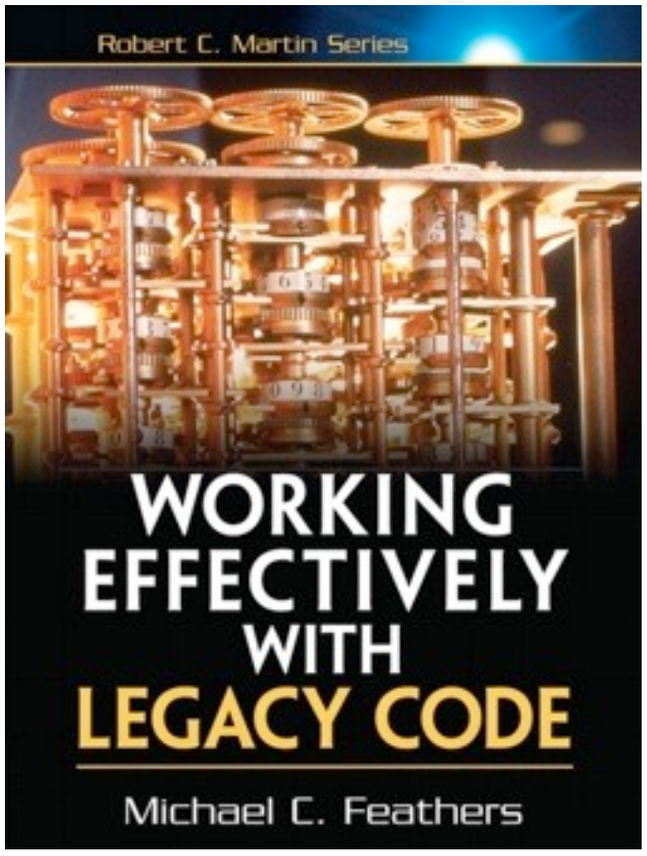
M3P

steve.freeman@m3p.co.uk

Other sources







M3P

steve.freeman@m3p.co.uk

Tacky Advert...