goto; conference aarhus

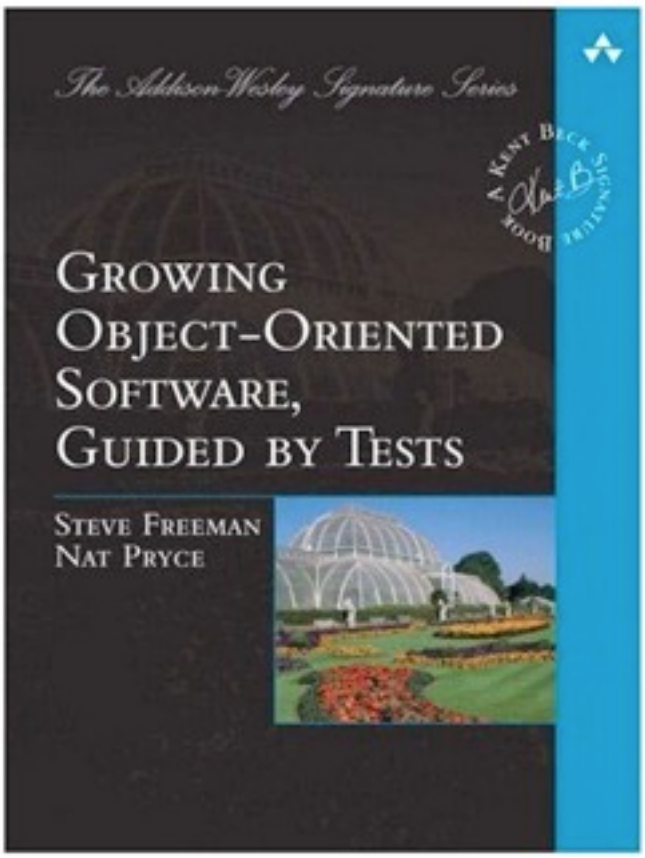
FRACTAL

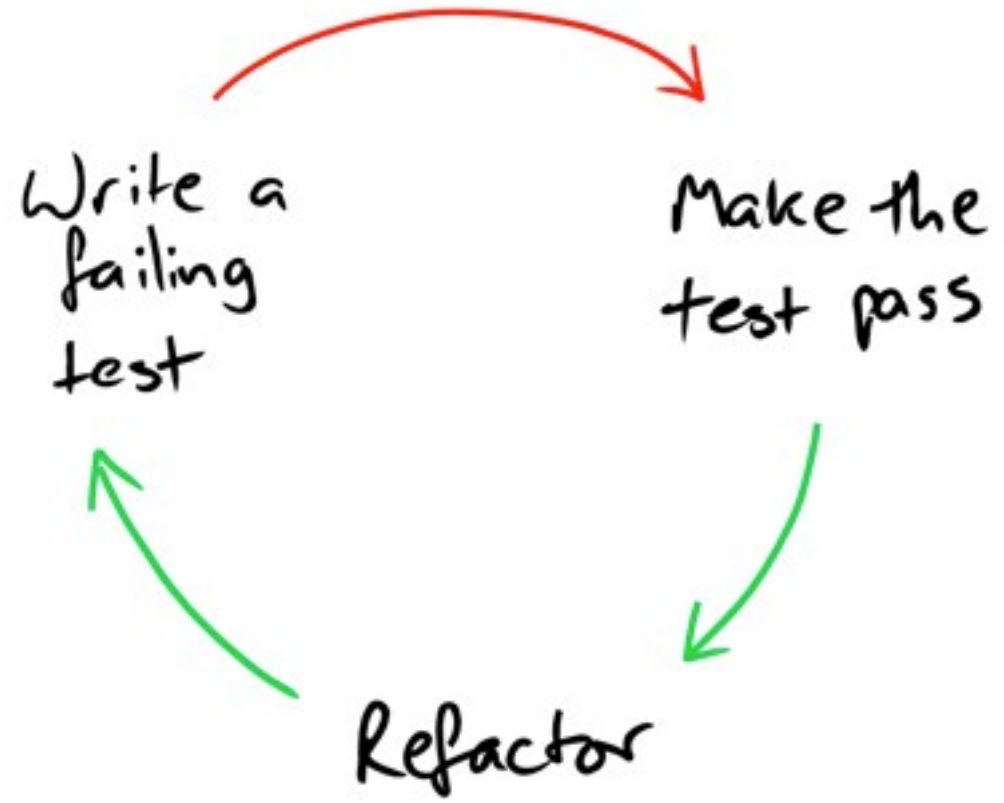
TEST-DRIVEN DEVELOPMENT

Steve Freeman
steve@higherorderlogic.com
@sf105

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com






```
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; }
        };

    }
```

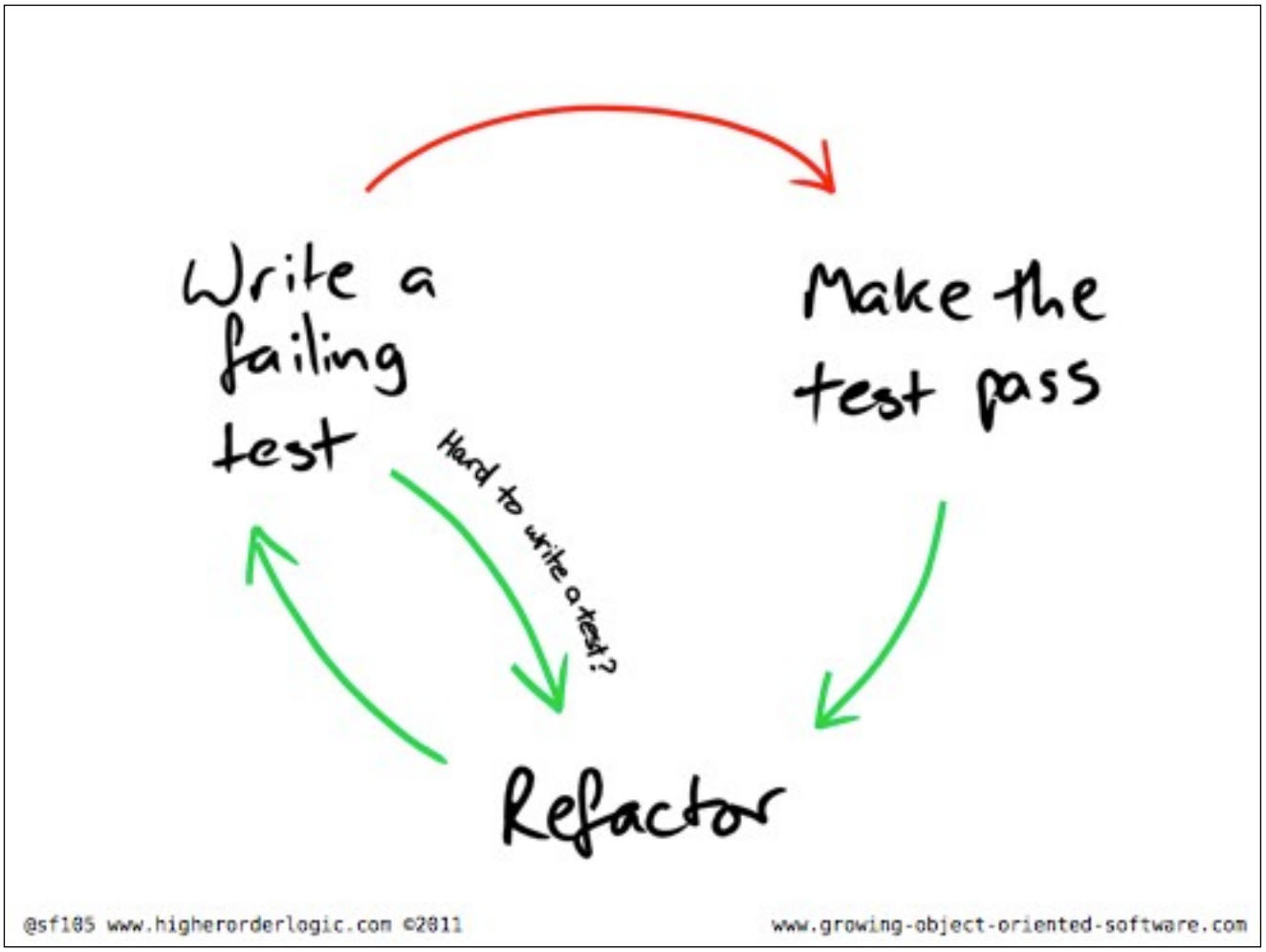
```
public void testInvariantAction() throws FrameworkException {
    mockGod = addMock(GodObject.class);
    expect(mockGod.getPriceDataManager()).andStubReturn(null);
    mockPersistenceManager = addMock(DatabaseFacade.class);
    registry = addMock(Registry.class);
    adminUser = new TestUser("Admin",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

    registry.finaliseThisThreadAs(String.class), isA(String.class));
    Date now = DateUtils.trimMinutesAndSecondsFrom(new Date());

    mockSystemVariableManager();
    mockLogger();
    mockCurrentPersistenceManager();
    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("prime", aCurrencyRate);
    out.put("otherprime", otherCurrencyRate);
    expect(mockPriceManager.getLatestExchangeRates(newList(aCurrencyPair, otherCurrencyPair))).andReturn(out);

    mockPMFactoryCleanup();

    replayMocks();

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



```
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);
}
```

```
    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();
    }
}
```

@sf185 www.higherorderlogic.com ©2011 www.growing-object-oriented-software.com



The test duplicates the code

```
@Test public void
duplicatesTheImplementation() {
  ReadValidateConvertAndSaveAction action =
    new ReadValidateConvertAndSaveAction(
      readHelper, validationHelper,
      convertHelper, repository);

  context.checking(new Expectations() {{
    one (readHelper).read();
                will(returnValue(inputRecord));
    one (validationHelper).validate(inputRecord);
    one (convertHelper).convert(inputRecord);
                will(returnValue(outputRecord));
    one (repository).save(outputRecord);
  }});

  action.doAction();
}
```

```
public class ReadValidateConvertAndSaveAction {
  public void doAction() {
    InputRecord inputRecord = readHelper.read();
    validationHelper.validate(inputRecord);
    OutputRecord outputRecord =
       convertHelper.convert(inputRecord);
    repository.save(outputRecord);
  }
}
```

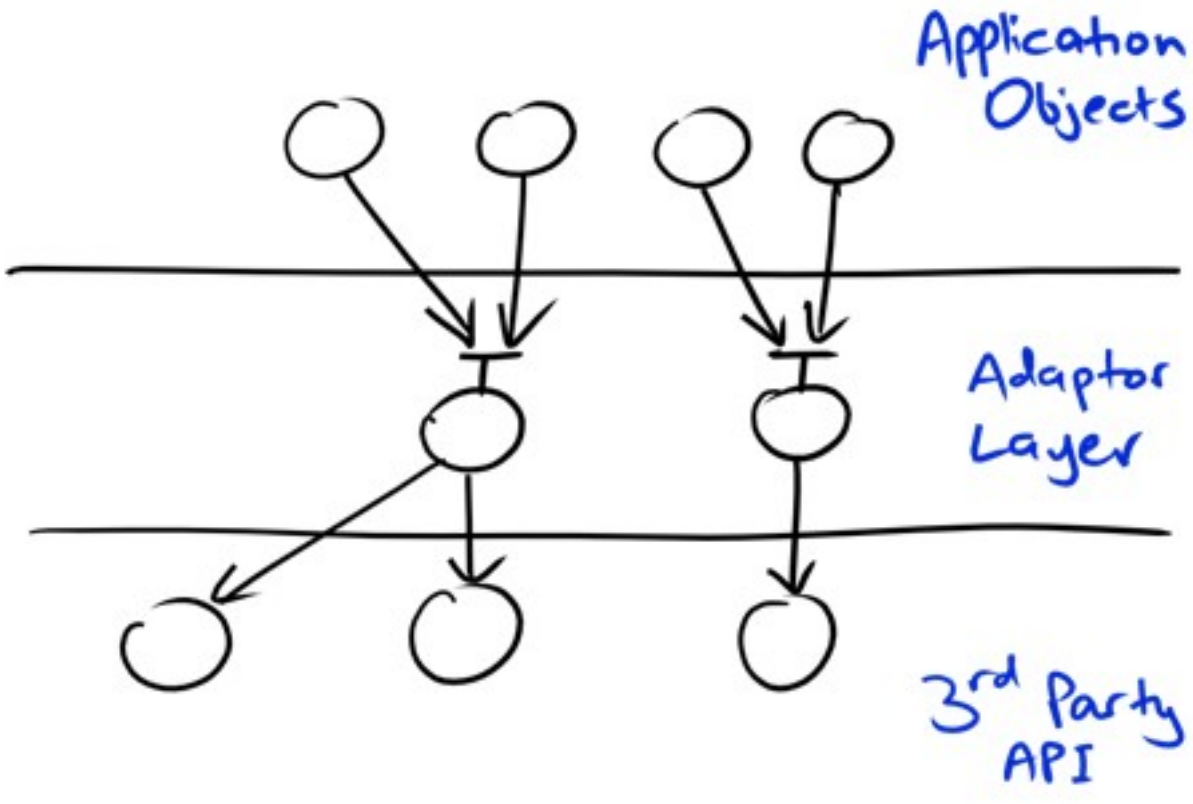
Too many assertions

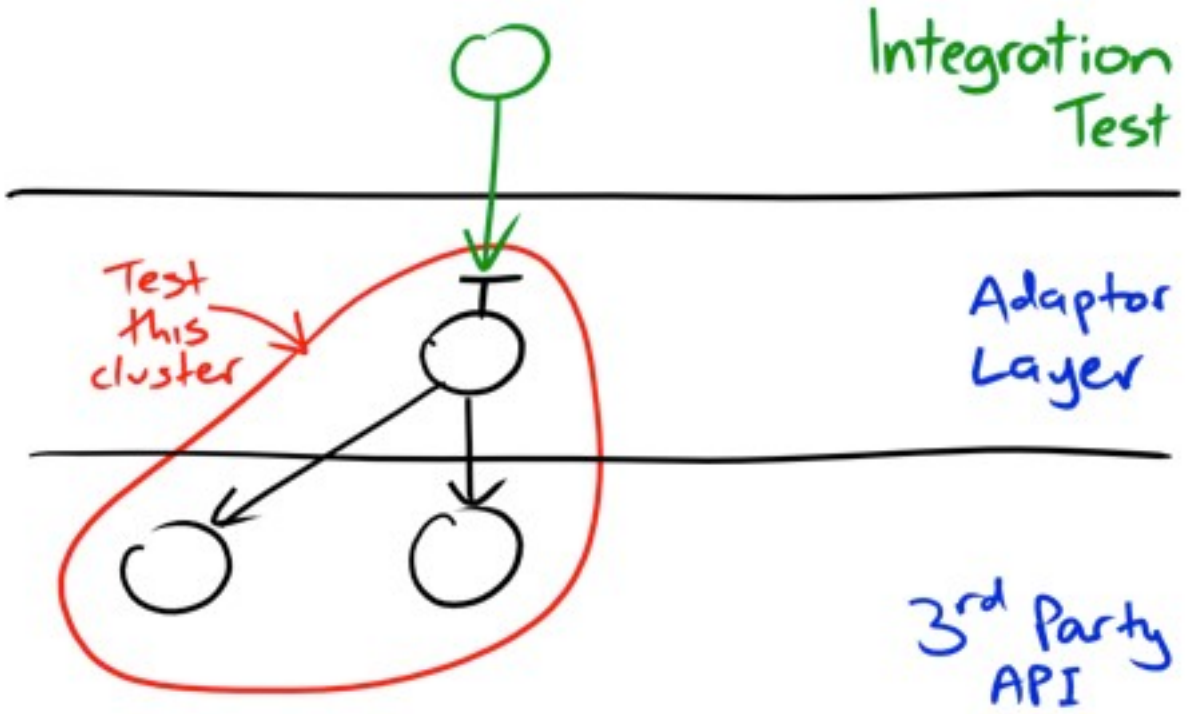
```
@Test public void decidesCasesWhenFirstPartyIsReady() {
  context.checking(new Expectations(){{
    allowing(firstPart).isReady(); will(returnValue(true));
    allowing(organiser).getAdjudicator(); will(returnValue(adjudicator));
    allowing(adjudicator).findCase(firstParty, issue); will(returnValue(case));

    one(thirdParty).proceedWith(case);
  }});

  claimsProcessor.adjudicateIfReady(thirdParty, issue);
}
```

```
public void adjudicateIfReady(ThirdParty thirdParty, Issue issue) {
  if (firstParty.isReady()) {
    Adjudicator adjudicator = organisation.getAdjudicator();
    Case case = adjudicator.findCase(firstParty, issue);
    thirdParty.proceedWith(case);
  } else{
    thirdParty.adjourn();
  }
}
```

```
public void adjudicateIfReady(ThirdParty thirdParty, Issue issue) {
  if (firstParty.isReady()) {
    thirdParty.startAdjudication(organisation, firstParty, issue);
  } else{
    thirdParty.adjourn();
  }
}
```

Faking the wrong objects





Test setup requires magic

```
@Test public void rejectsRequestsNotWithinTheSameDay()
{
  receiver.acceptRequest(FIRST_REQUEST);
  // the next day
  assertFalse("too late now",
              receiver.acceptRequest(SECOND_REQUEST));
}
```

```
public boolean acceptRequest(Request request) {
  final Date now = new Date();
  if (dateOfFirstRequest == null) {
    dateOfFirstRequest = now;
   } else if (firstDateIsDifferentFrom(now)) {
    return false;
  }
  // process the request
  return true;
}
```

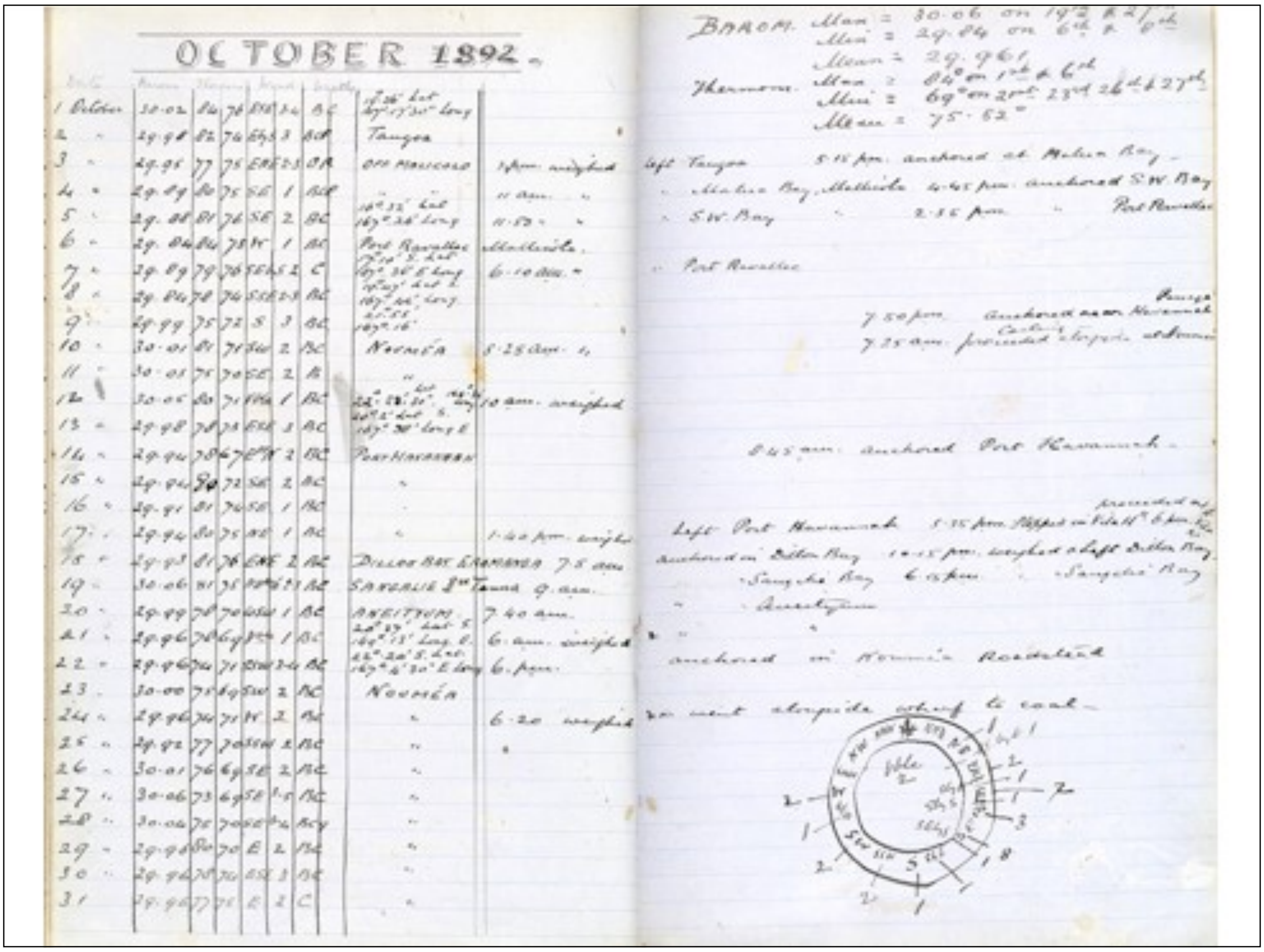
```
Date now = new Date();
```

```
@Test public void rejectsRequestsNotWithinTheSameDay() {
  Receiver receiver = new Receiver(stubClock);
  stubClock.setNextDate(TODAY);
  receiver.acceptRequest(FIRST_REQUEST);

  stubClock.setNextDate(TOMORROW);
  assertFalse("too late now",
              receiver.acceptRequest(SECOND_REQUEST));
}
```

```
public boolean acceptRequest(Request request) {
  final Date now = clock.now();
  if (dateOfFirstRequest == null) {
   dateOfFirstRequest = now;
  } else if (firstDateIsDifferentFrom(now)) {
   return false;
  }
  // process the request
  return true;
}
```

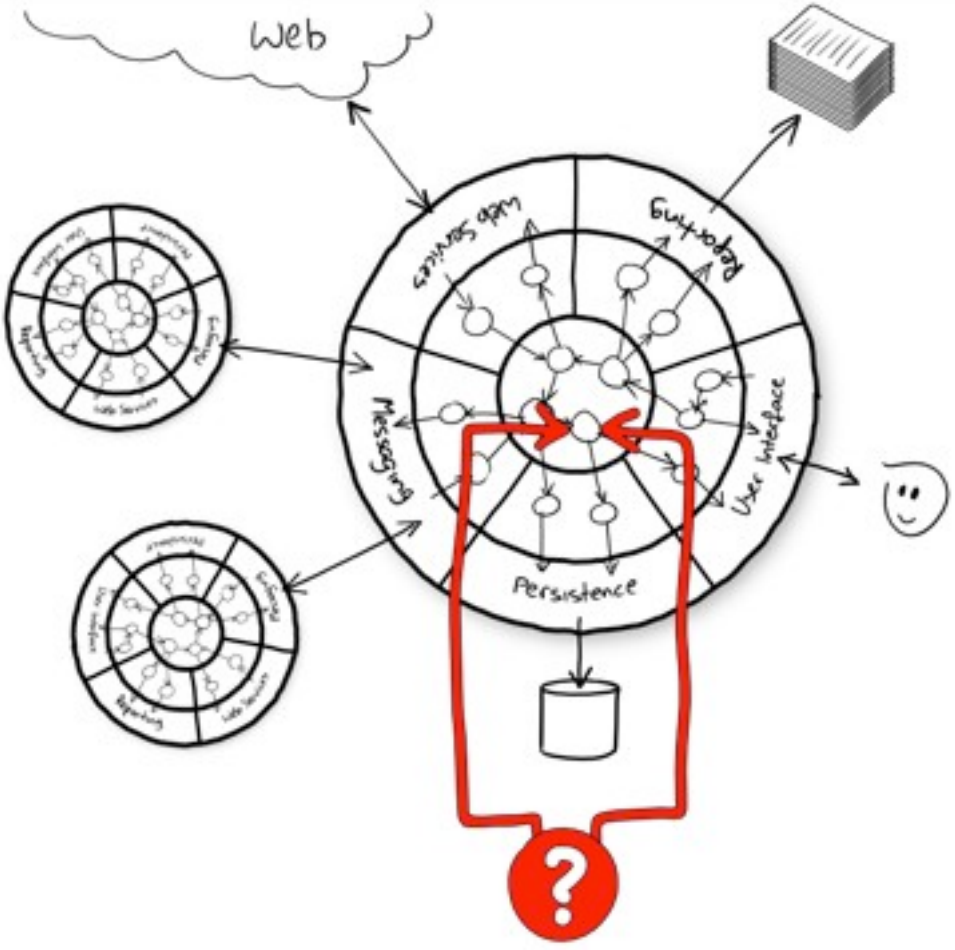
```
@Test public void rejectsRequestsOutsideAllowedPeriod() {
  Receiver receiver = new Receiver(expiryChecker);
  context.checking(new Expectations() {{
    allowing(expiryChecker).hasExpired();
                           will(returnValue(false));
  }});

  assertFalse("too late now", receiver.acceptRequest(REQUEST));
}
```

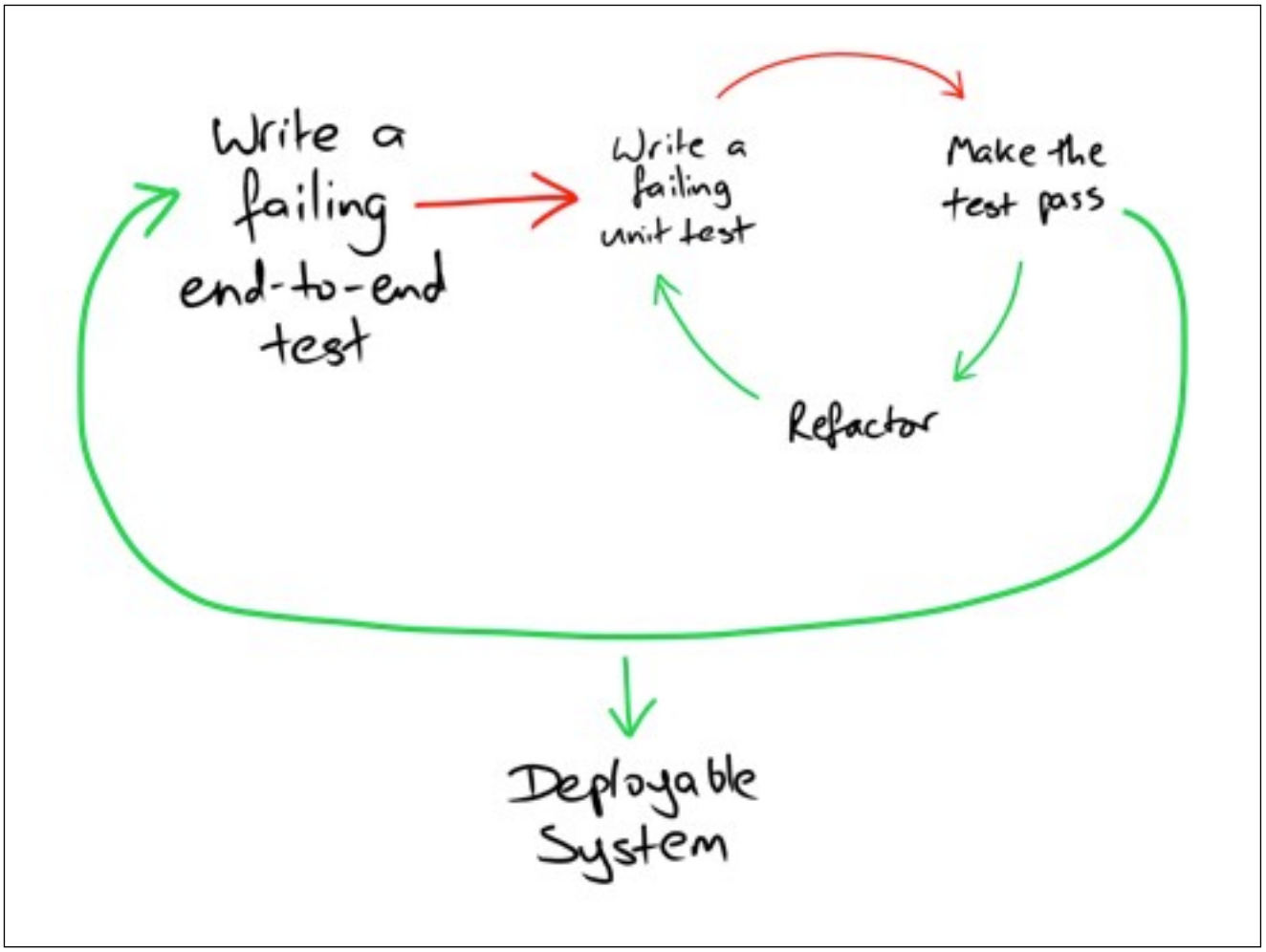
```
public boolean acceptRequest(Request request)
{
  if (expiryChecker.hasExpired()) {
   return false;
  }
  // process the request
  return true;
}
```

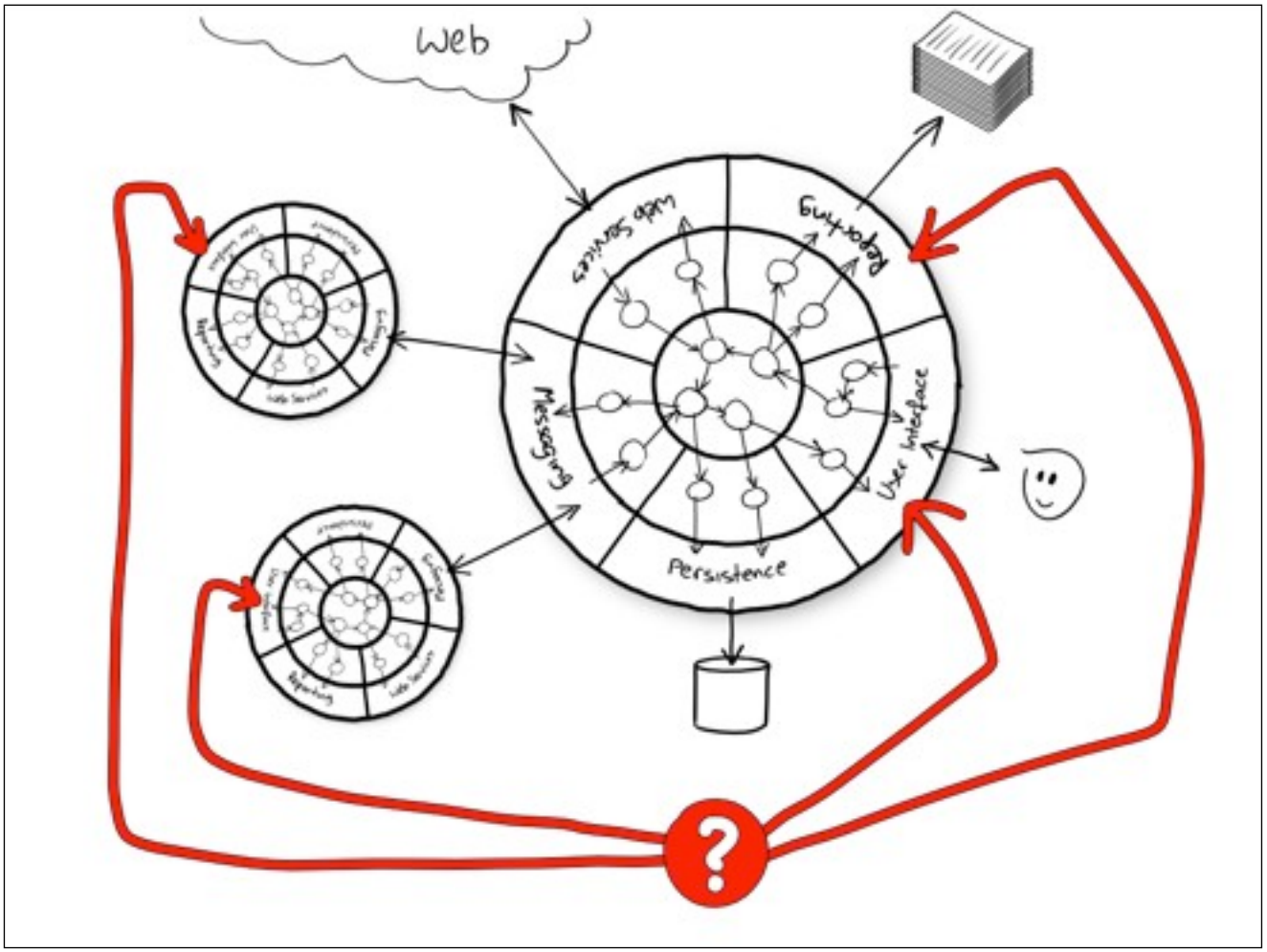



```java
public class UsefulComponent {
  static Log log =
    LogFactory.getLog(UsefulComponent.class);

  public void performMiracle() {
    if (! fullMoon()) {
      log.error("Needs extra incantation");
    }
    ...
```

```
public class UsefulComponent {
  private Wizard wizard;

  public void performMiracle() {
    if (! fullMoon()) {
      wizard.reportMissingIncantation();
    }
    ...
```




Write a failing end-to-end test
Write a failing unit test
Make the test pass
Refactor
Deployable System


Web
Web Services
Reporting
Messaging
User Interface
Persistence
?

To test a system we need to


know what the system is doing

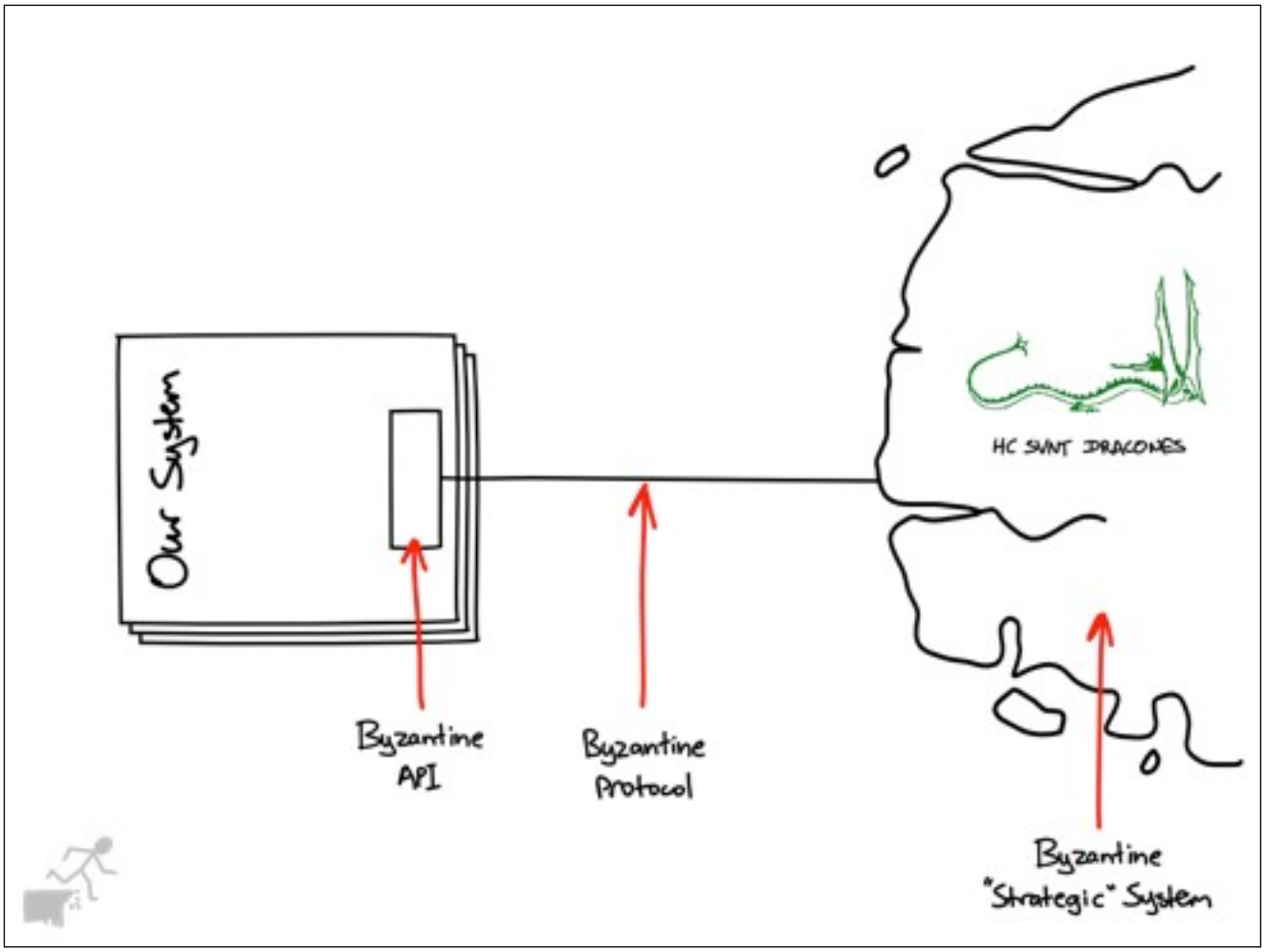

know when it has stopped doing it

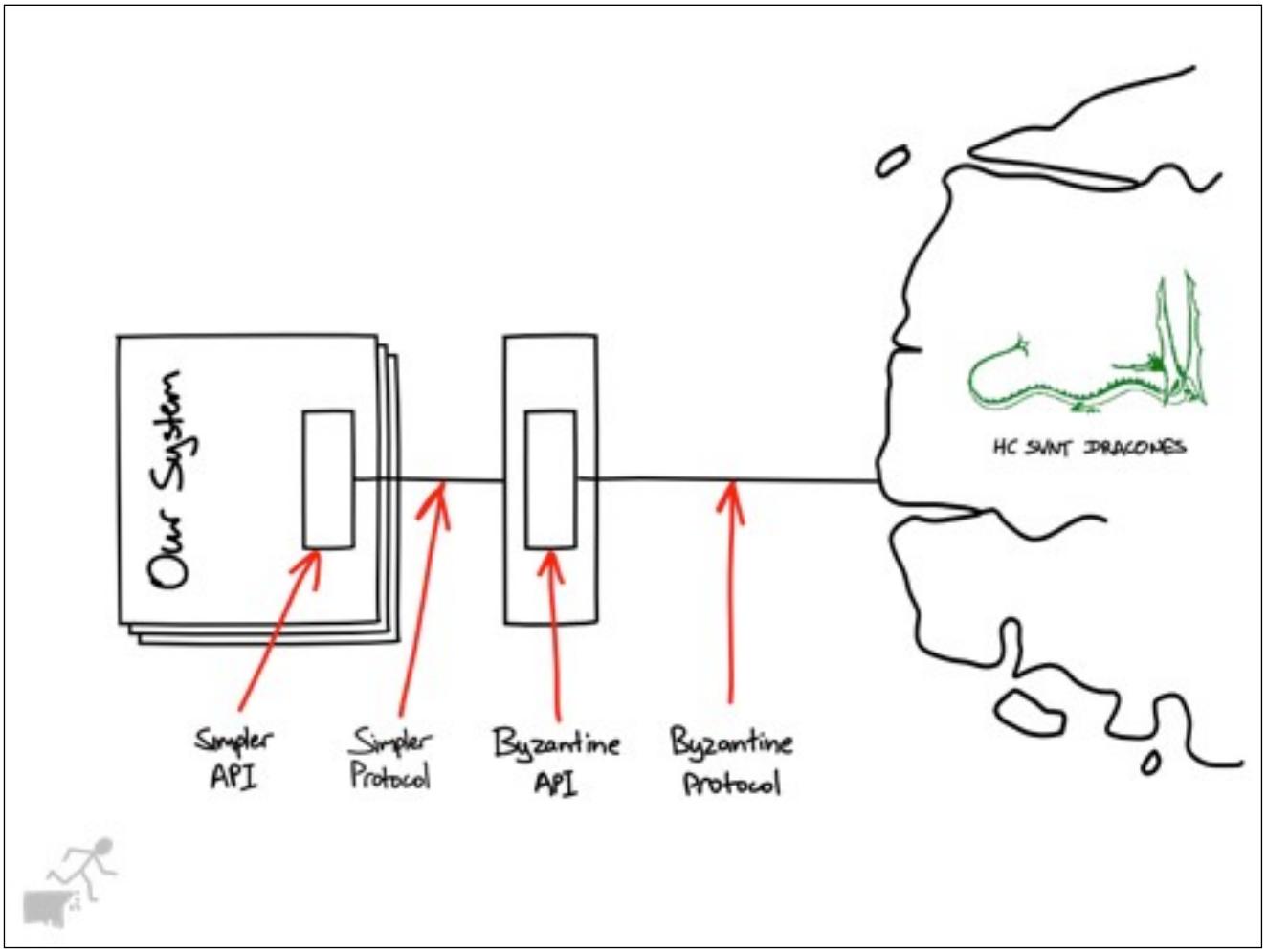

know when it has gone wrong

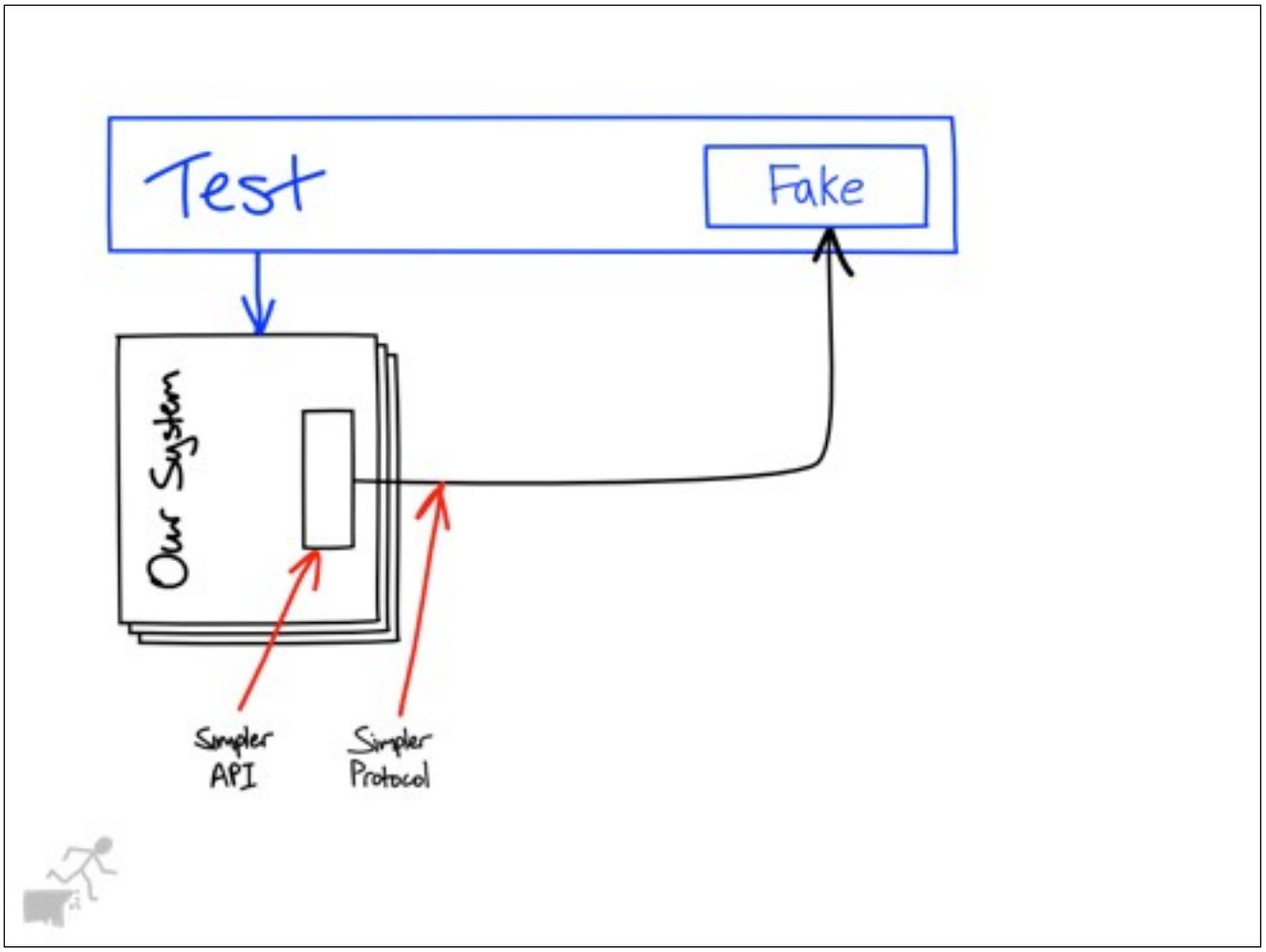

determine why it has gone wrong

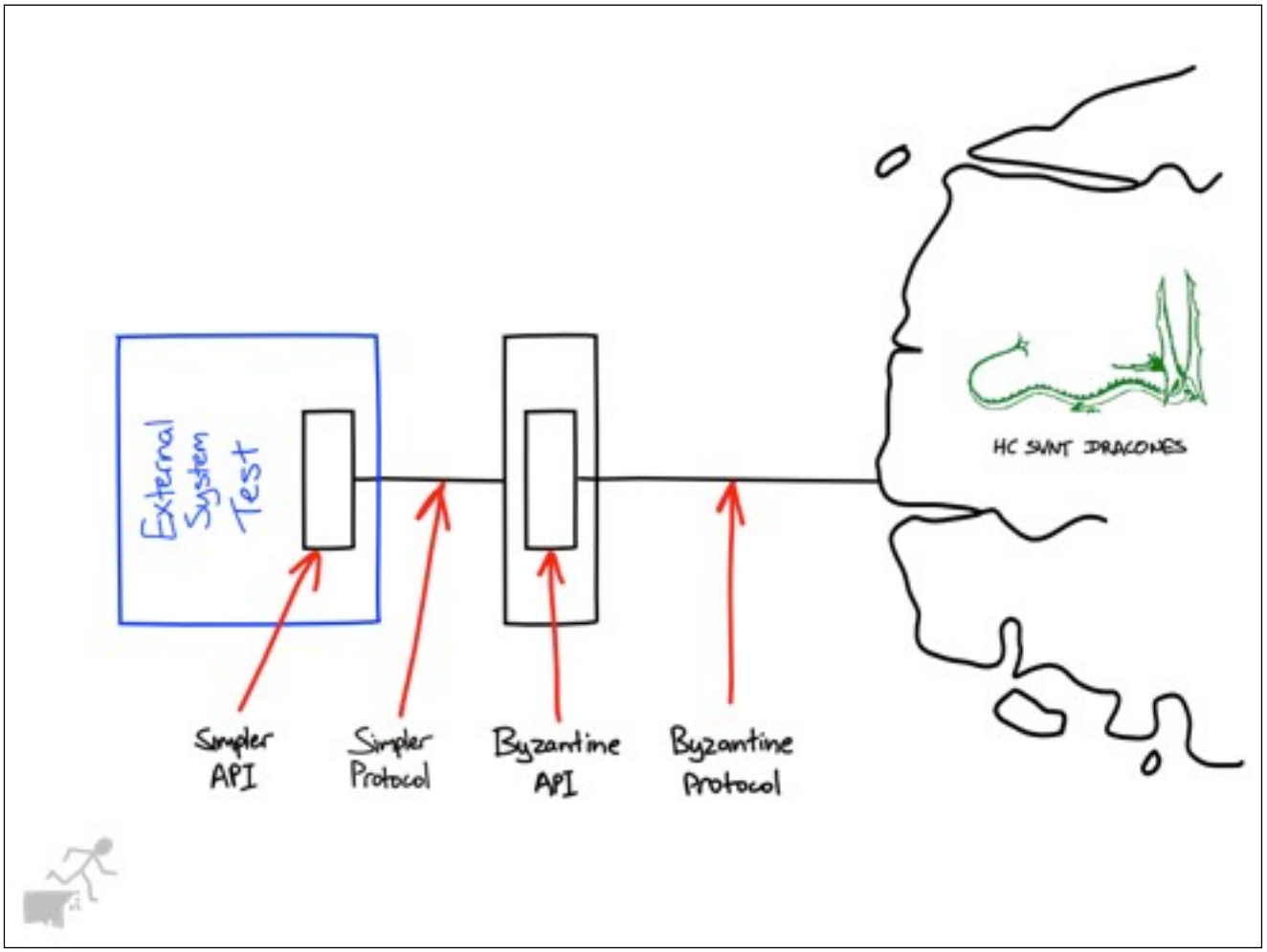

restore it to a good state

Testing at the system's edges


Our System
HC SVNT DRACONES
Byzantine API
Byzantine Protocol
Byzantine "Strategic" System


Our System
HC SVNT DRACONES
Simpler API
Simpler Protocol
Byzantine API
Byzantine Protocol


Test
Fake
Our System
Simpler API
Simpler Protocol


External System Test
HC SVNT DRACONES
Simpler API
Simpler Protocol
Byzantine API
Byzantine Protocol

Logorrhoea

```
try {
    String cookie = null;
    CookieJar curCook = getNecessaryCookie(false);
    if(curCook != null) {
      cookie = curCook.toString();
      LOG.debug("have a cookie");
    }

    LOG.info("Loading page: " + auctionURL);

    loadedPage = Http.receivePage(
        Http.makeRequest(auctionURL, cookie));

    LOG.info("page " + auctionURL + " loaded ok");

} catch(FileNotFoundException fnfe) {
    LOG.error("Item not found: " + auctionURL);
    throw fnfe;
}
```

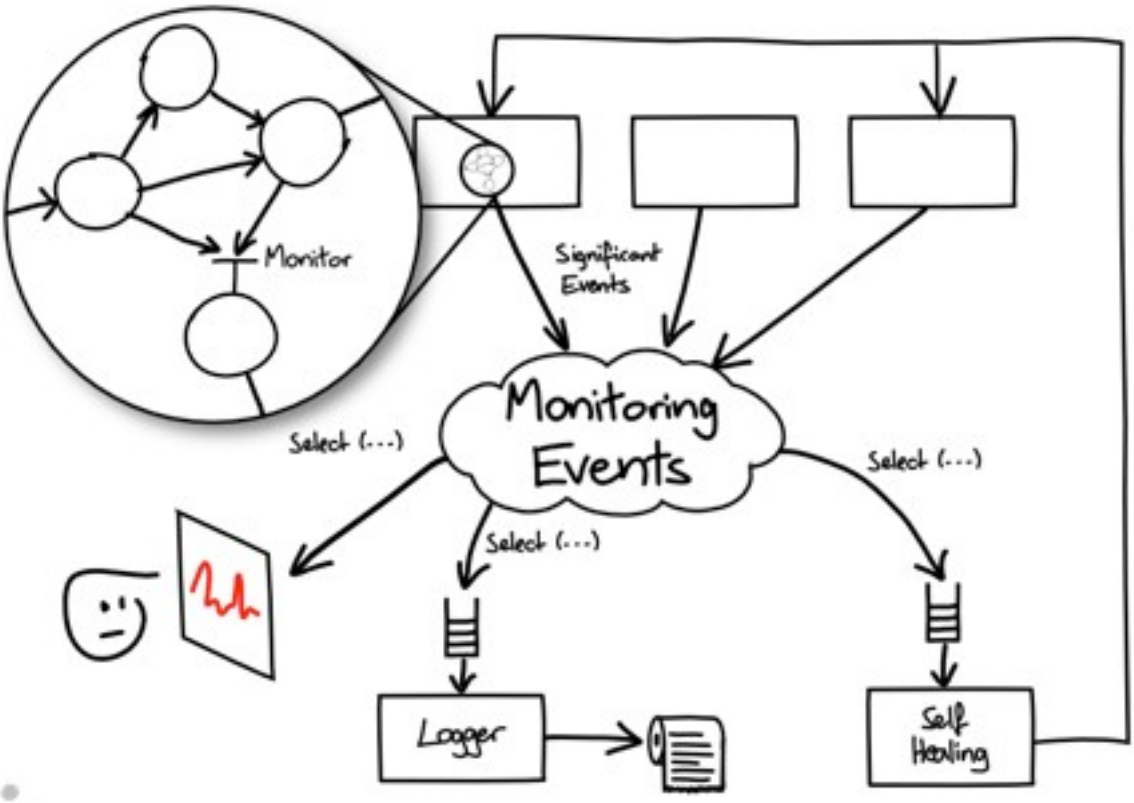
Messy Code

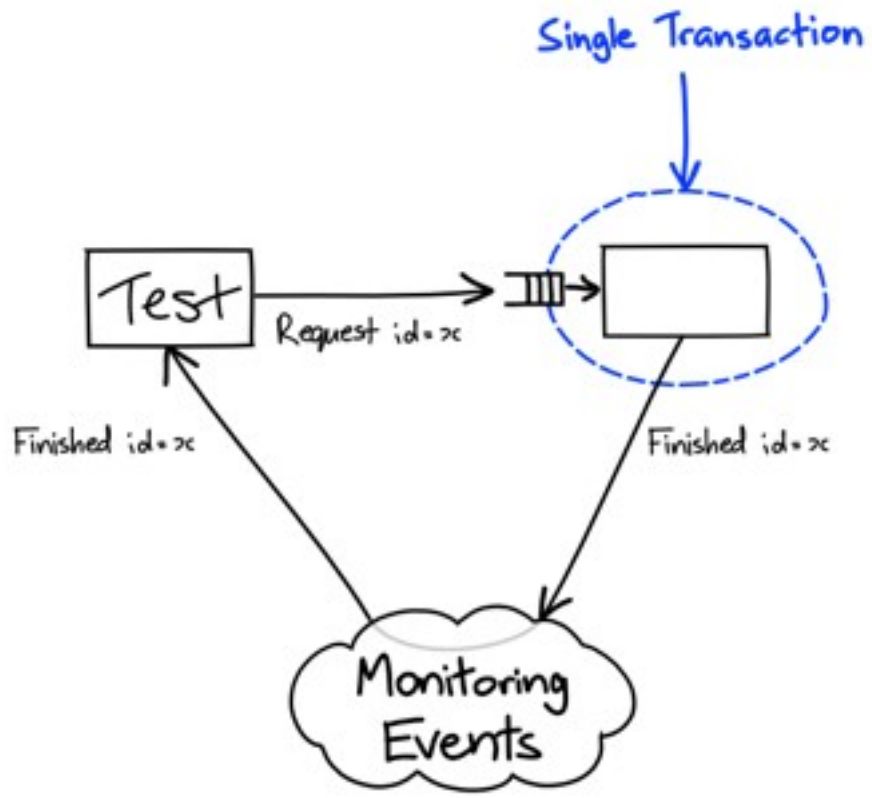
Inconsistent log levels

Inconsistent formats

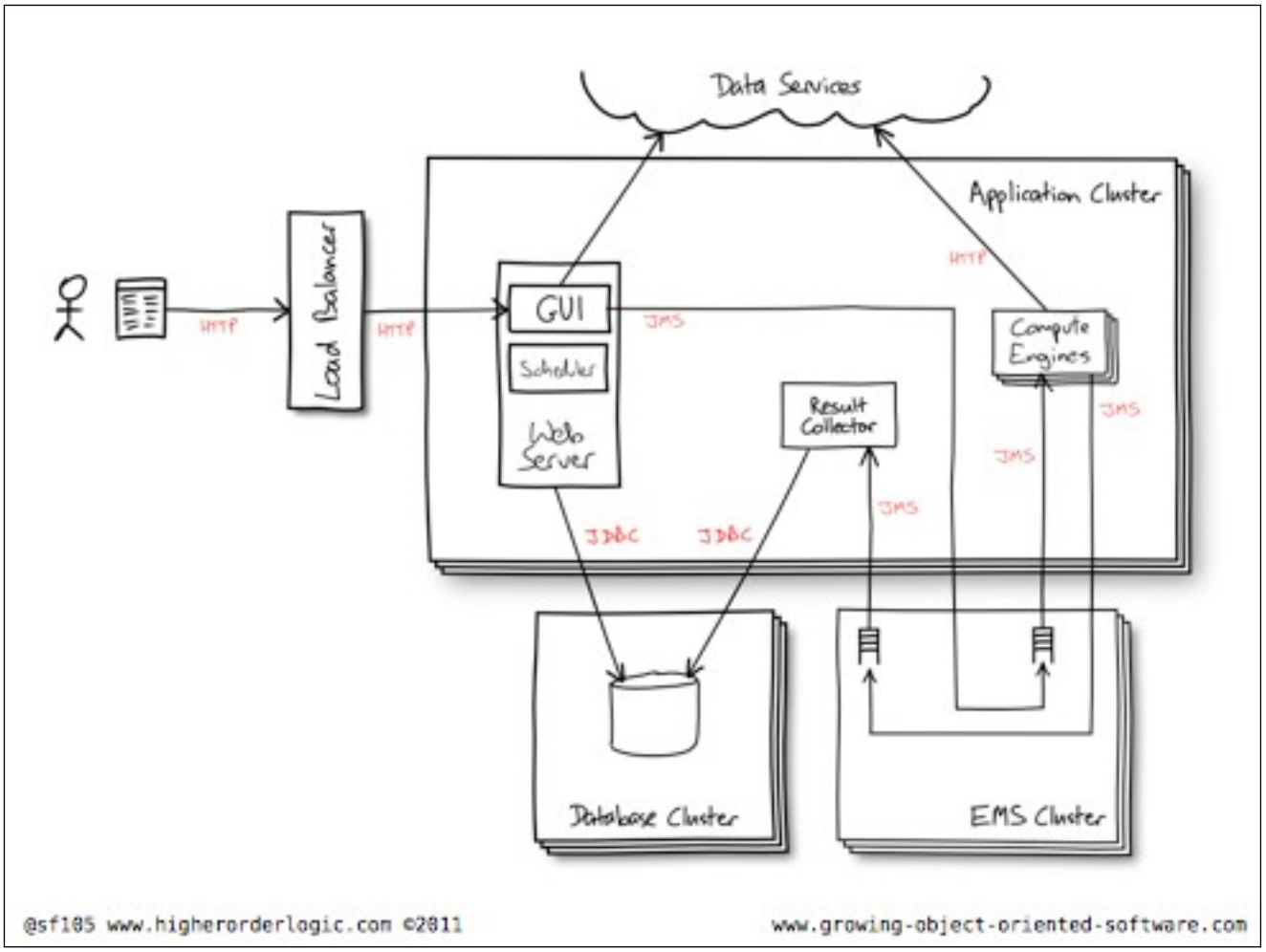
Duplicated reports

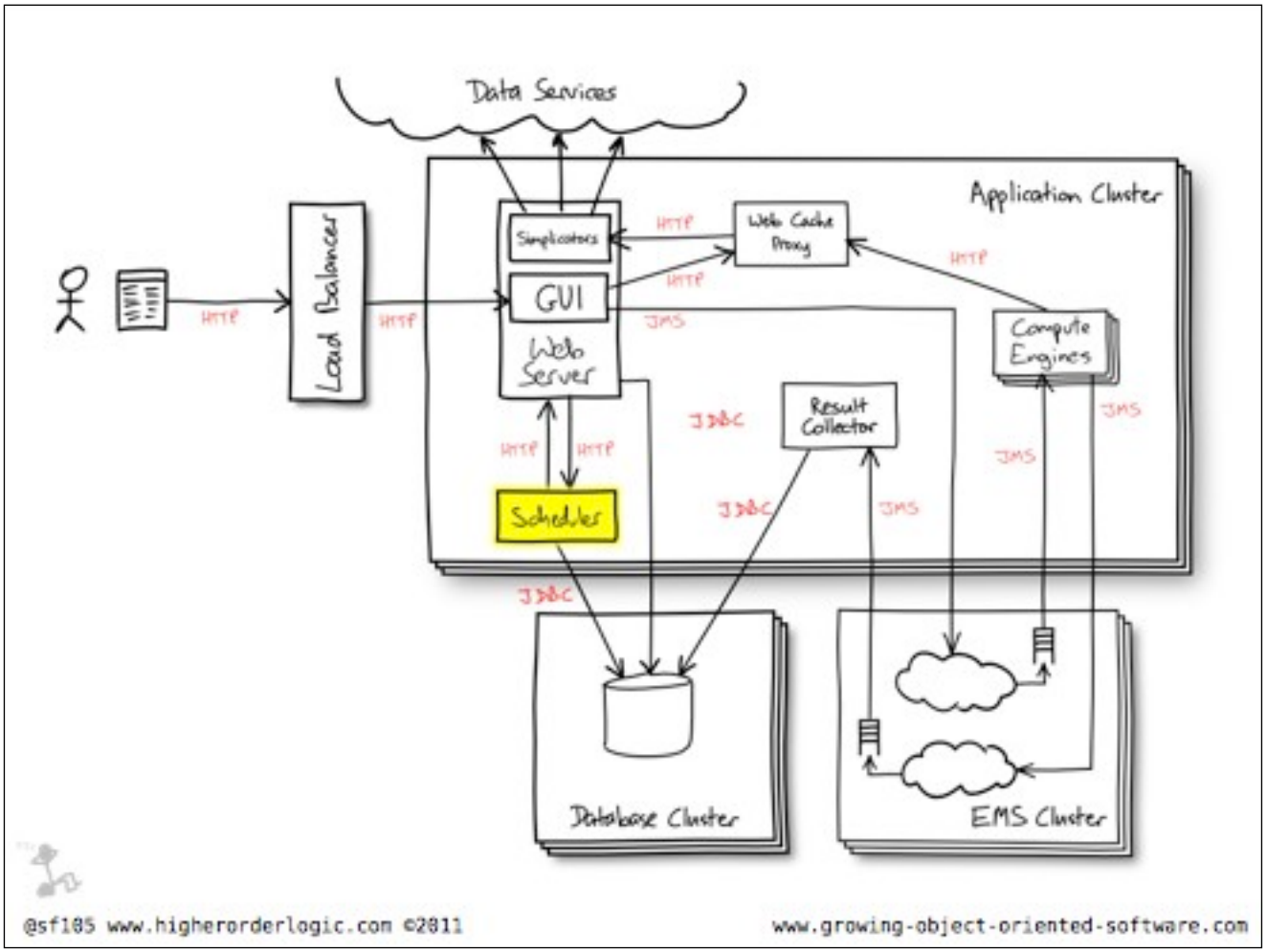
```
String cookie = null;
CookieJar curCook = getNecessaryCookie(false);
if(curCook != null) {
  cookie = curCook.toString();
}

monitor.requestReceived(auctionURL, cookie);

loadedPage = Http.receivePage(
    Http.makeRequest(auctionURL, cookie));

monitor.loadedPageFor(auctionURL);
```

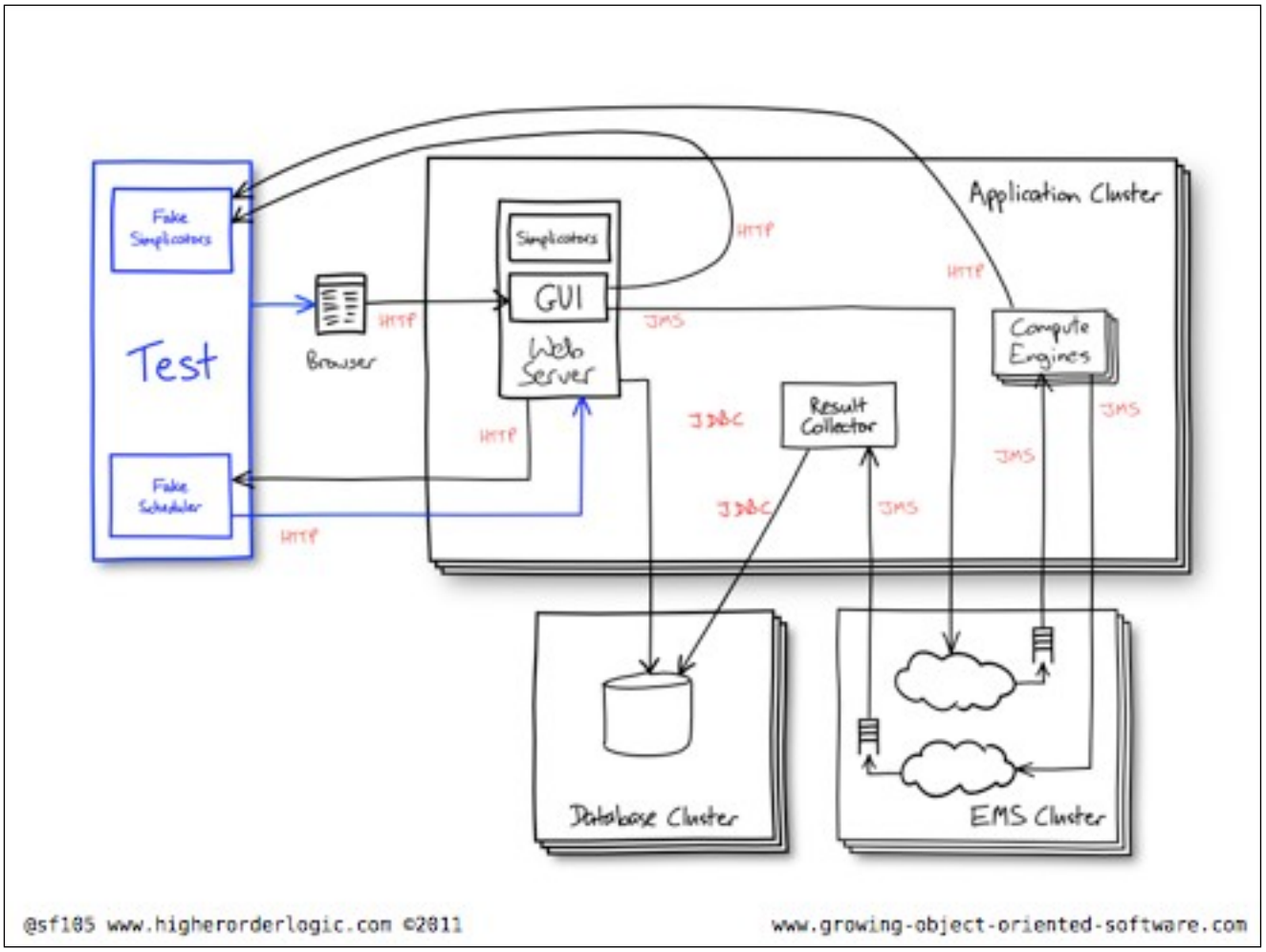


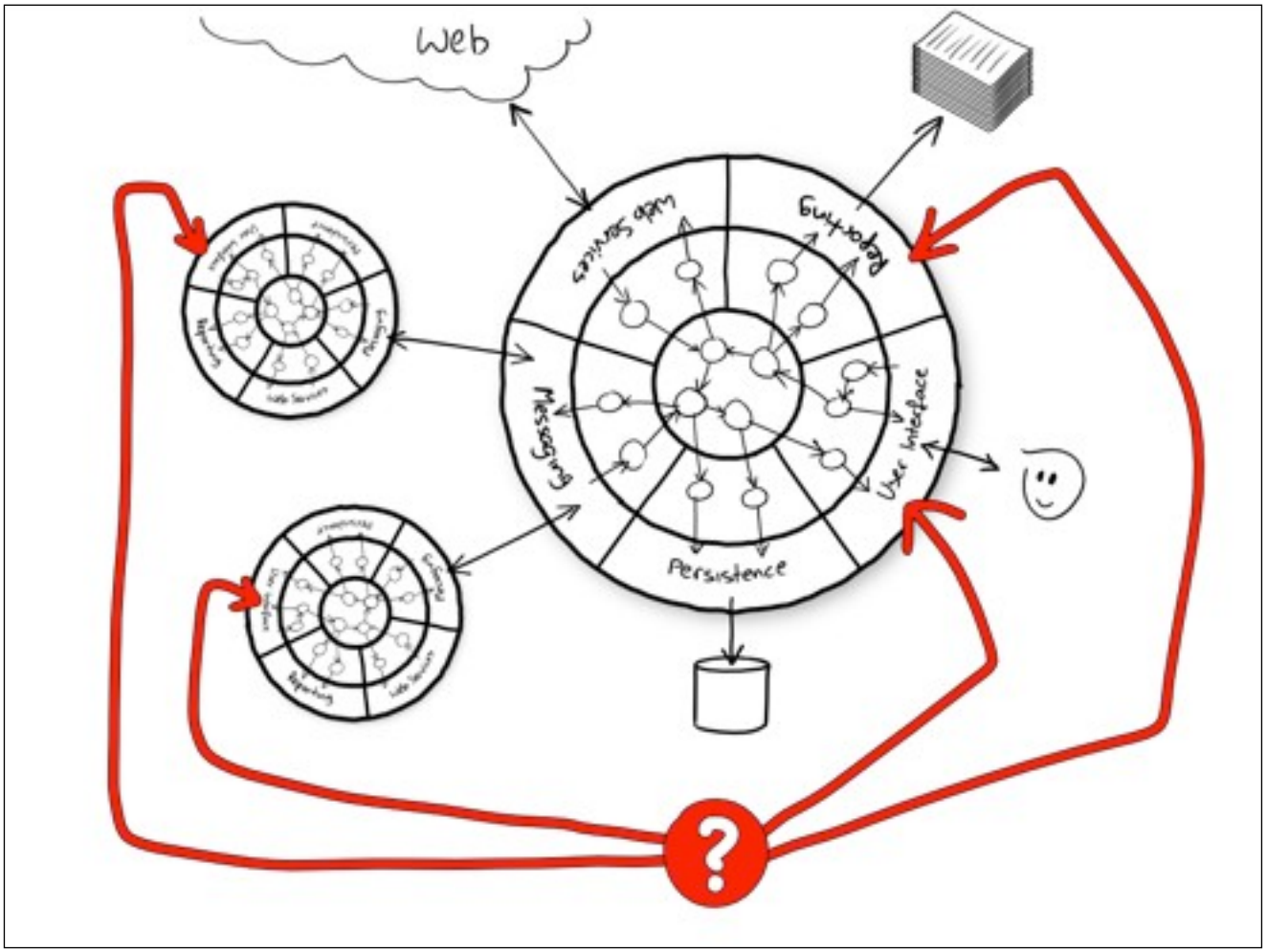


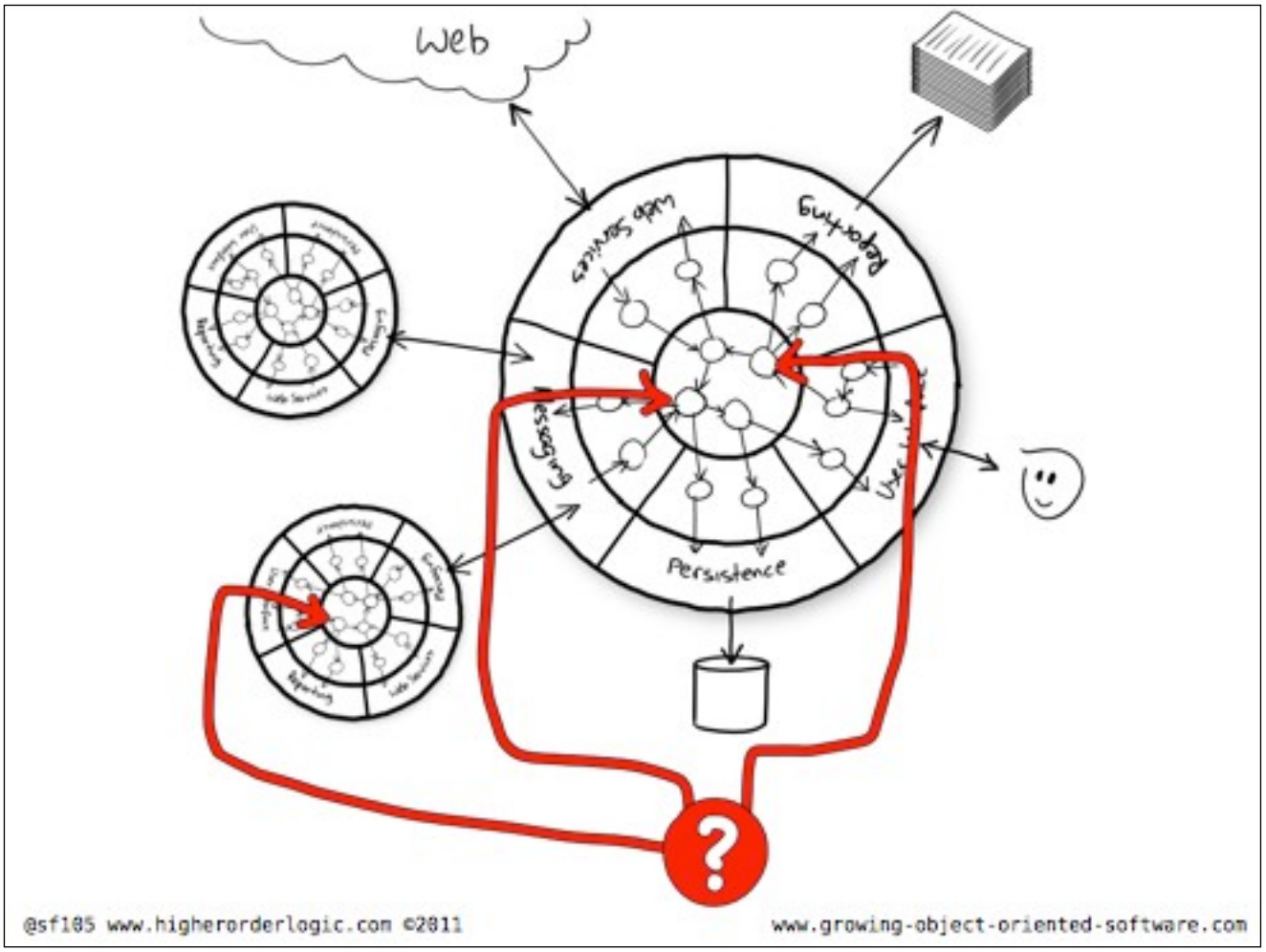
Spontaneous system behaviour

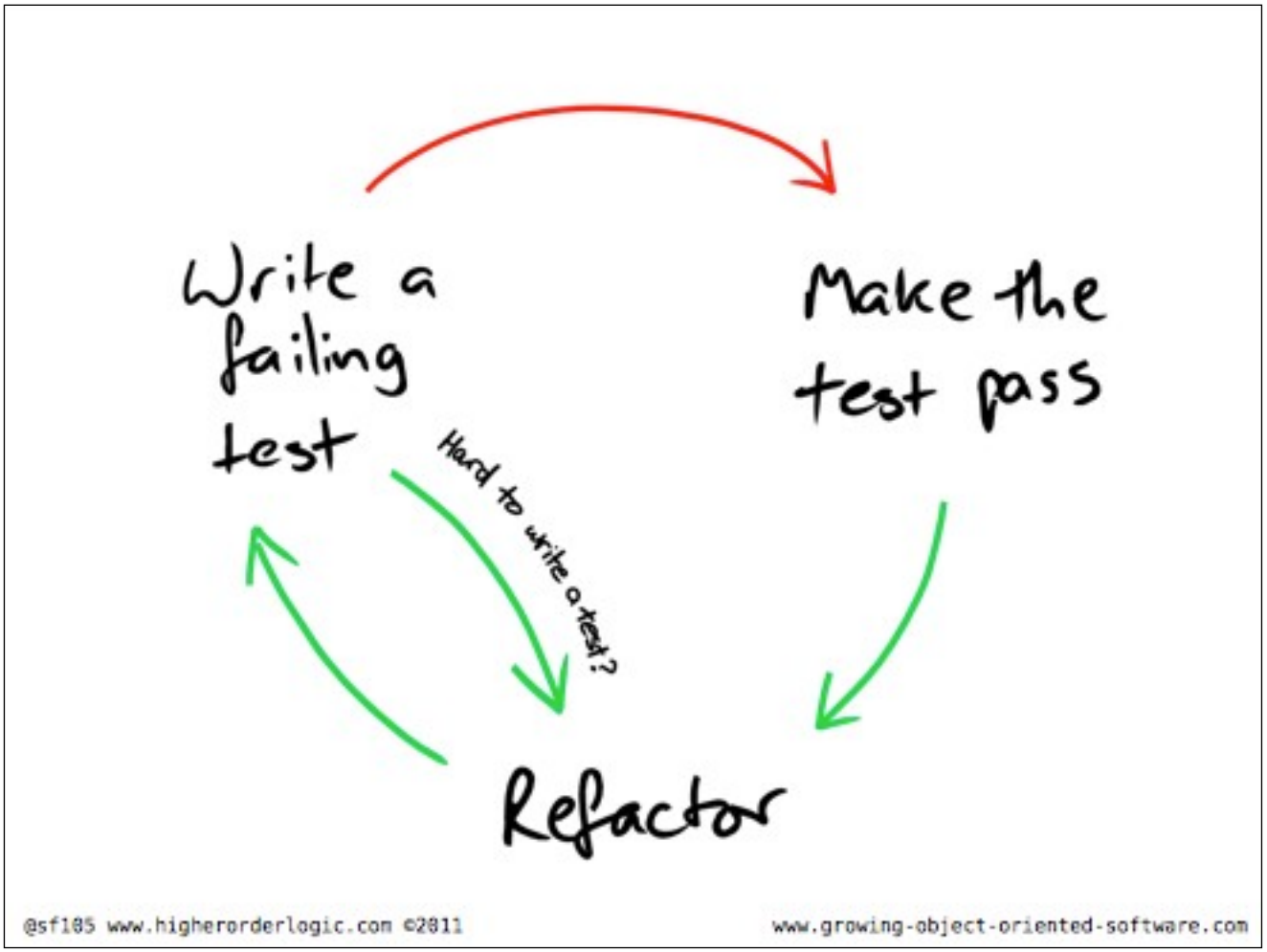

Data Services
Application Cluster
Load Balancer
HTTP
GUI
Scheduler
Web Server
JMS
Compute Engines
Result Collector
JDBC
Database Cluster
EMS Cluster
@sf185 www.higherorderlogic.com ©2011
www.growing-object-oriented-software.com


Data Services
Application Cluster
Load Balancer
HTTP
Simplecasters
Web Cache Proxy
GUI
Web Server
JMS
Compute Engines
Result Collector
JDBC
Scheduler
Database Cluster
EMS Cluster
@sf185 www.higherorderlogic.com ©2011
www.growing-object-oriented-software.com


Fake Simplicators
Test
Fake Scheduler
Browser
HTTP
Simplicators
GUI
Web Server
Application Cluster
Compute Engines
Result Collector
JDBC
JMS
Database Cluster
EMS Cluster
@sf105 www.higherorderlogic.com ©2011
www.growing-object-oriented-software.com


Web
Web Services
Deployment
Messaging
User Interface
Persistence
?


Web
Web Services
Reporting
Persistence
?
@sf185 www.higherorderlogic.com ©2011
www.growing-object-oriented-software.com


Write a failing test
Make the test pass
Hard to write a test?
Refactor
@sf185 www.higherorderlogic.com ©2011
www.growing-object-oriented-software.com

To ~~test~~ support a system we need to

know what the system is doing

know when it has stopped doing it

know when it has gone wrong

determine why it has gone wrong

restore it to a good state

Benefit of TDD

Unit → System easier to modify

System → System easier to support

goto; conference aarhus

FRACTAL TEST-DRIVEN DEVELOPMENT

Steve Freeman
steve@higherorderlogic.com
@sf105

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com