Unit Testing Fundamentals

Richard Paul
Kiwiplan NZ Ltd
27 Feb 2009

What is Unit Testing?

You all know what unit testing is, but here is a definition:

In computer programming, unit testing is a method of testing that verifies the individual units of source code are working properly...
 -- *http://en.wikipedia.org/wiki/Unit_testing*

Benefits of Unit Testing

Automated regression tests, ensure code continues to work

Executable, evolving documentation of how the code works.

Allows code to be less brittle allowing for refactoring without fear.

Test code in isolation of the rest of the system (which may not be written yet).

Fewer bugs!

Unit Testing vs Functional Testing

Unit tests are written from a programmer's perspective. They ensure that a particular method of a class successfully performs a set of specific tasks.

Functional tests are written from a user's perspective. These tests confirm that the system does what users are expecting it to.

-- http://www.ibm.com/developerworks/library/j-test.html

Structure of a Unit Test

- Test method per test case.
- Each test should run in isolation.

```
public class ItemControllerTest {

  private ItemController itemController;
  @Mock private ItemService itemService;
  private Map<String, Object> modelMap;

  @Before
  public void setUp() {
    itemController = new ItemController(itemService);
    modelMap = new HashMap<String, Object>();
  }

  @Test
  public void testViewItem() throws Exception {
    Item item = new Item(1, "Item 1");
    when(itemService.getItem(item.getId())).thenReturn(item);

    String view = itemController.viewItem(item.getId(), modelMap);

    assertEquals(item, modelMap.get("item"));
    assertEquals("viewItem", view);
  }
```

Unit Testing Best Practices

Unit tests should run fast.

- Allows developers to code fast.
- Don't hit the database unless you really need to.

Tests should assert only what is required to exercise the code.

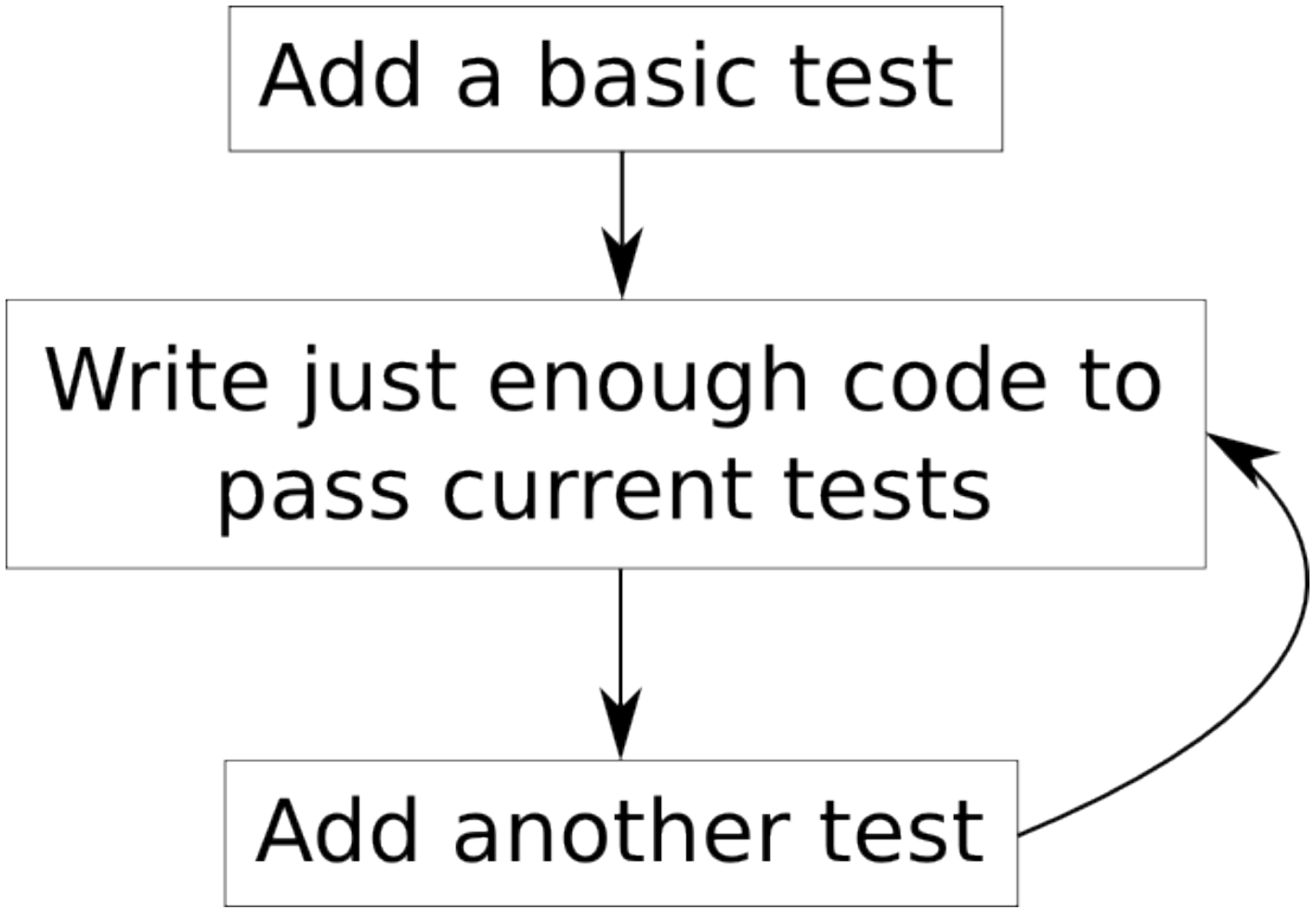
- Over-specification leads to excessive test maintenance.

Unit tests should be developed in parallel to the code.

Unit tests should be run automatically every time code is checked in.

  - Using a continuous integration machine e.g.Hudson.
  - Provides quick feedback for any integration problems.

Test Driven Development (TDD)



TDD Advantages

You are encouraged to think about how your object is to be used.

Develop faster, no compile/deploy cycle to check code works.

Debugging a unit test is much simpler than a deployed application.

Set of regression tests allowing you to refactor as you develop without fear of breaking previous functionality.

Unit tests are likely to cover more edge cases as the object model is fresh in your mind.

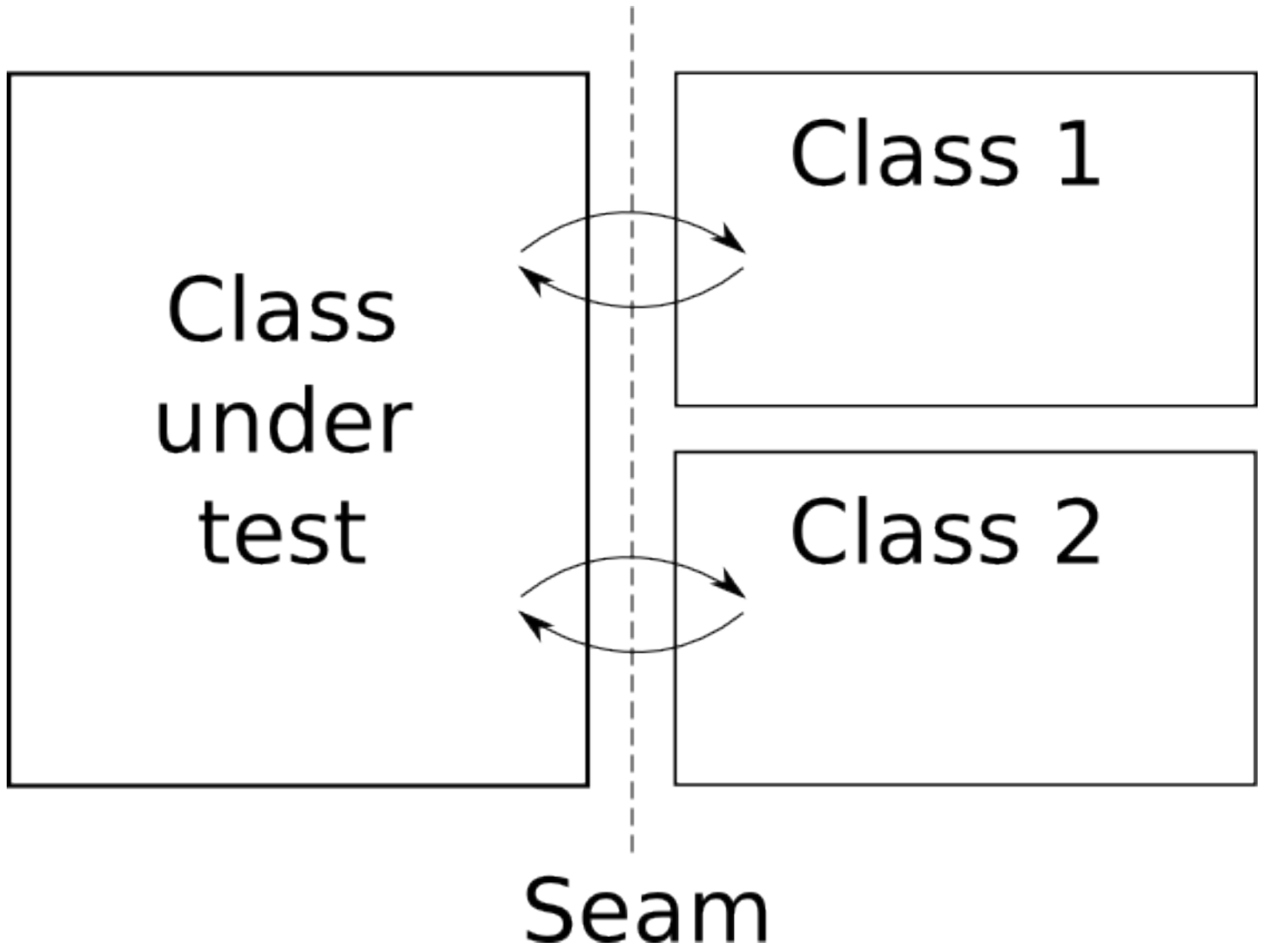
Expose a Seam

A seam is a line at which the code being tested can be isolated.

Collaborators on the other side of the seam should be replaceable to ensure we can isolate the code we wish to test.

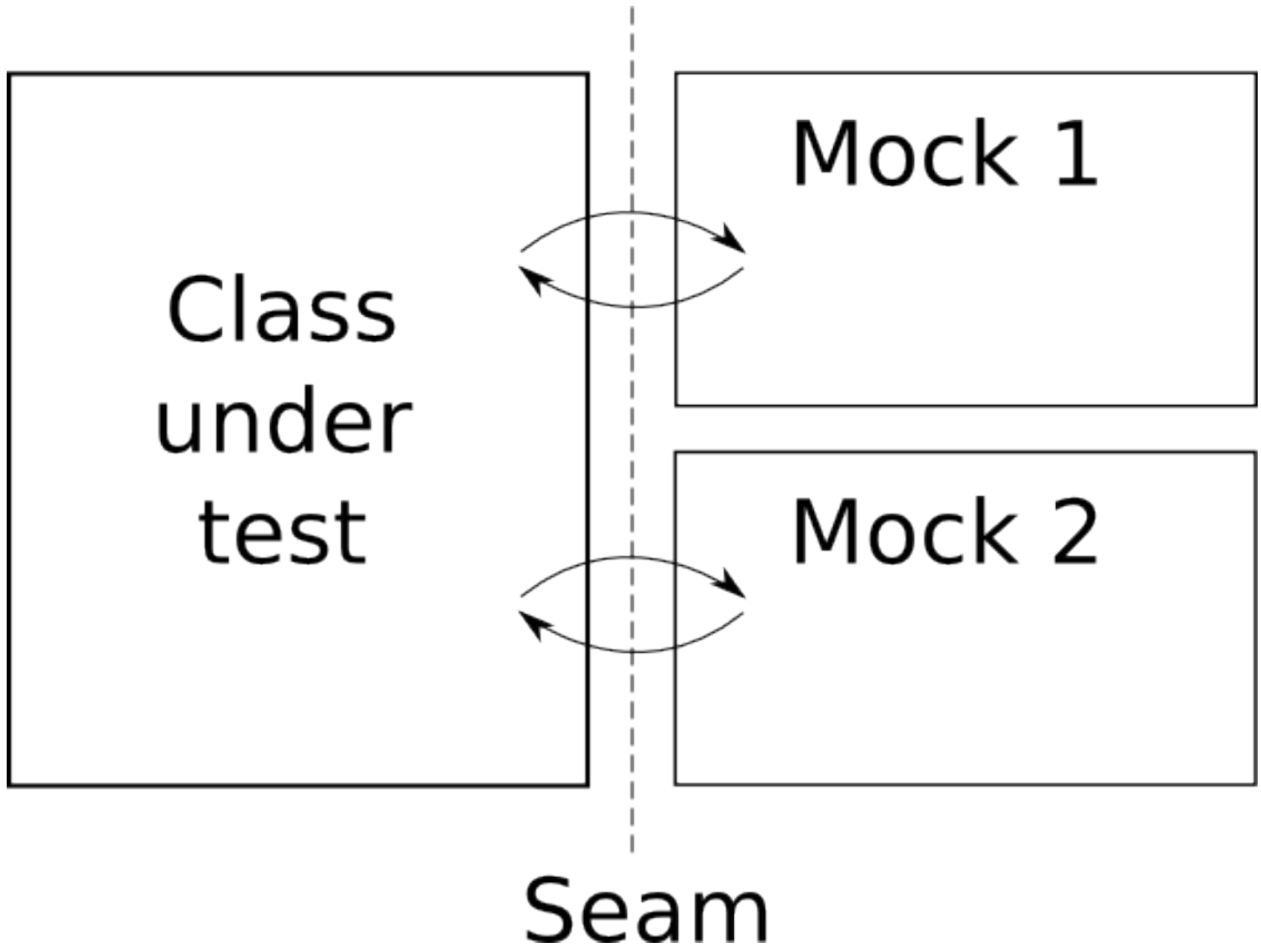
Code that exposes a seam is generally loosely coupled.

Exposing a Seam



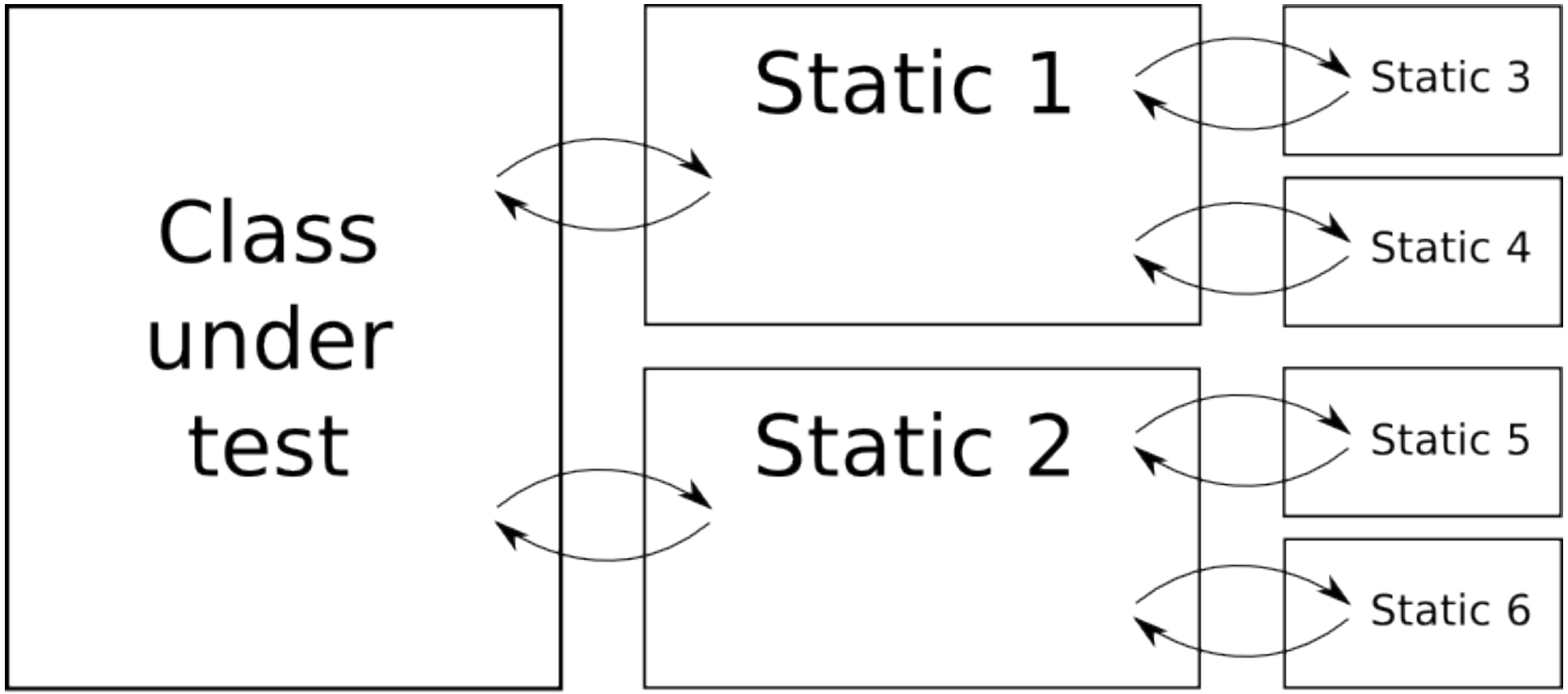
At runtime, production classes are used.

Mocking out collaborators



Replaced production classes with mocks for testing.

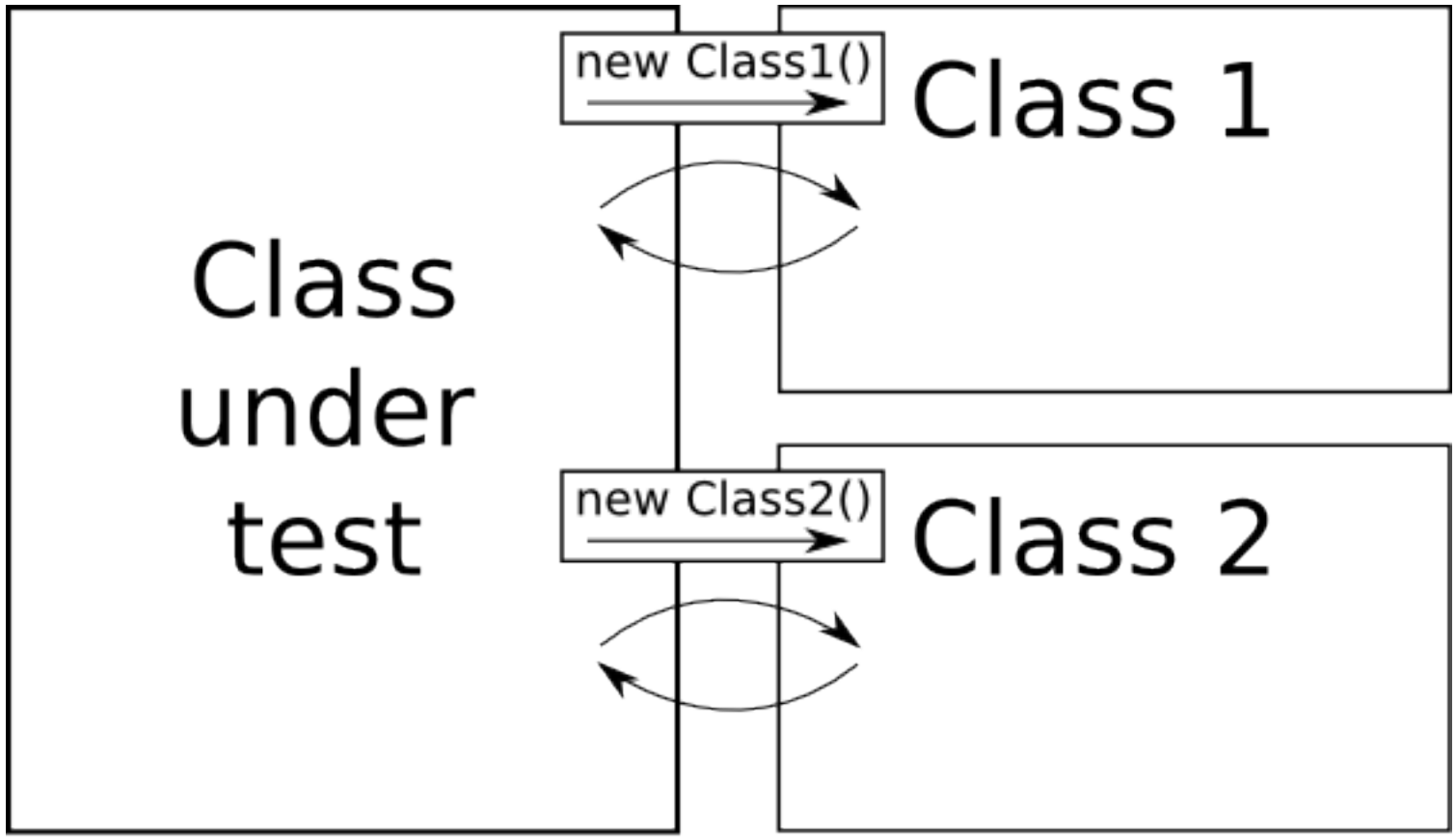
Anti-patterns - Static Methods



No seam

Use of static methods means there is no way to replace the collaborators in the test, resulting in overly complex tests.

Locally instantiated collaborators



No seam

The class under test instantiates Class 1 and Class 2. We can't replace these classes in the test so we have no seam.

Anti-pattern Further Reading

Static Methods are Death to Testability
http://misko.hevery.com/2008/12/15/static-methods-are-death-to-testability/

To "new" or not to "new"…
http://misko.hevery.com/2008/09/30/to-new-or-not-to-new/

You may have noticed, Miško Hevery is my hero.

Mocking Frameworks

Mocking frameworks provide a convenient means for ensuring unit tests remained focused by providing the ability to 'mock out' other objects.

Frameworks include:
Java

- Mockito
- EasyMock
- JMock

.Net

- Moq (requires .Net 3.5)
- Rhino
- TypeMock

Example - Accessing a Collaborator

In order to provide a seam for code that uses services there needs to exist a way to swap out the service used.

Dependency Injection

- Java
  - Spring Framework
  - Guice
- .Net
  - Springframework.Net
  - Castle MicroKernel/Windsor

Wikipedia has a huge list of existing frameworks.

Service Access - Poor mans dependency injection

Example - Dependency Injection

Required resources are injected into the class that requires them.

Constructor Injection

```
public class ItemController {

  private ItemService itemService;

  public ItemController(ItemService itemService) {
    this.itemService = itemService;
  }
```

Setter Injection

```
public class ItemController {

  private ItemService itemService;

  public ItemController() {}

  public void setItemService(ItemService itemService) {
    this.itemService = itemService;
  }
```

Example - Service Access

If you don't have access to a dependency injection container, some of the testing benefits can be gained using a singleton approach.

```
public class ServiceAccess {

  private static ItemService itemService;

  public static ItemService getItemService() {
    return itemService;
  }

  public static void setItemService(ItemService itemService) {
    ServiceAccess.itemService = itemService;
  }

}
```

In your test setup you can then set itemService to a mocked version to expose the seam.

Live Example

Java: Eclipse, JUnit 4.5, Mockito 1.7

VB.Net: Visual Studio 2005, NUnit 2.4, Rhino mocks 3.5

For more Mockito (Java) examples see:
http://www.rapaul.com/2008/11/19/mocking-in-java-with-mockito/

Test Coverage

Keep the bar green AND the coverage green.

Java

- EMMA - has an eclipse plugin EclEmma

```
public String viewItem(long id, Map<String, Object> modelMap) {
  try {
    Item item = itemService.getItem(id);
    modelMap.put("item", item);
    return "viewItem";
  } catch (ItemNotFoundException e) {
    modelMap.put("exception", e);
    return "redirect:/errorView";
  }
}
```

.Net

- NCover - commercial, has Visual Studio support

Summary

Each test case should be clearly defined in a separate test.

Tests provide executable documentation of how code works.

Test maintainability is important as tests make up a large portion of the code base.

- Document the use case the test covers.
- Over-specification leads to more maintenance.

Exposing a seam keeps code loosely coupled and simplifies testing through the use of mock objects.

Test Driven Development focuses development on how the code is used and reduces build/deploy iterations = Faster.