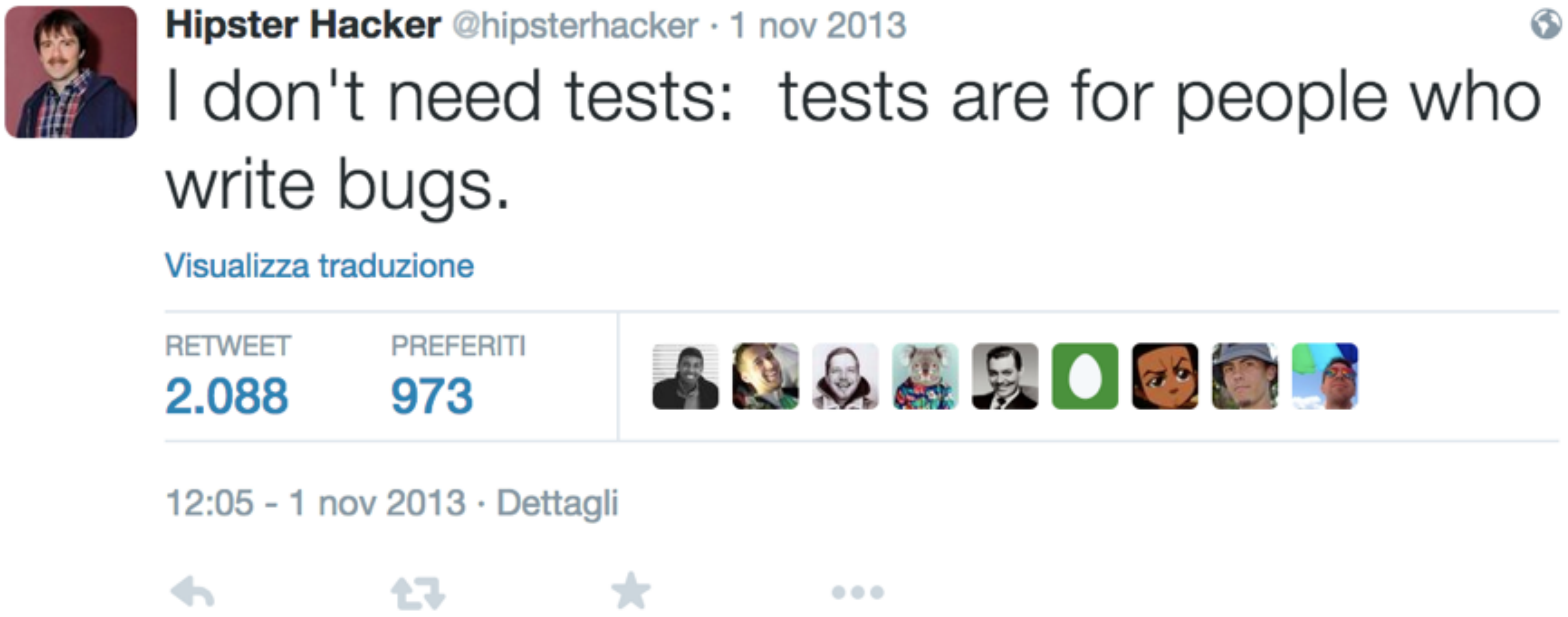
Fabio Collini

Testing



Hipster Hacker @hipsterhacker · 1 nov 2013

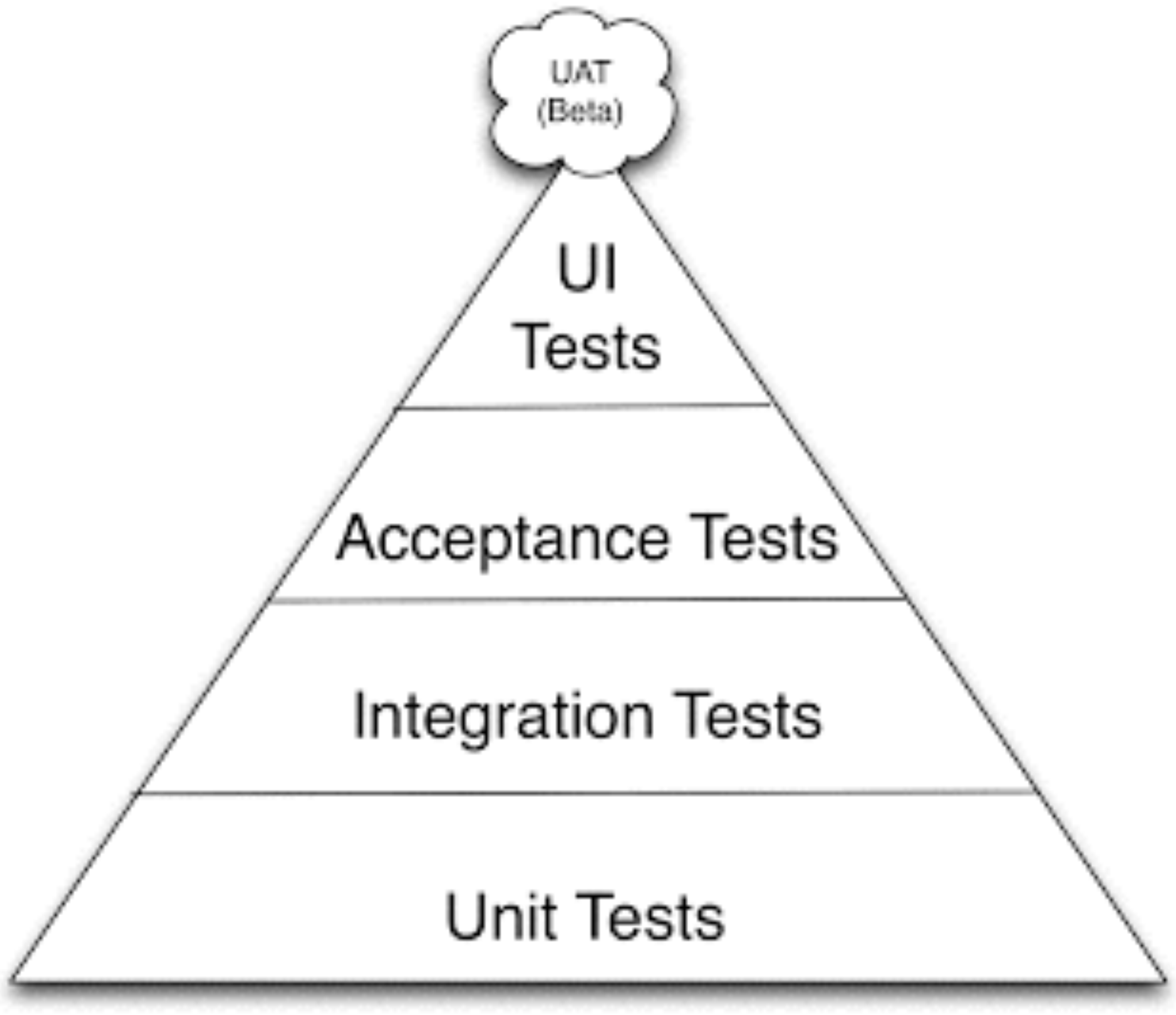
I don't need tests: tests are for people who write bugs.

Visualizza traduzione

RETWEET	PREFERITI
2.088	973

12:05 - 1 nov 2013 · Dettagli

Testing pyramid

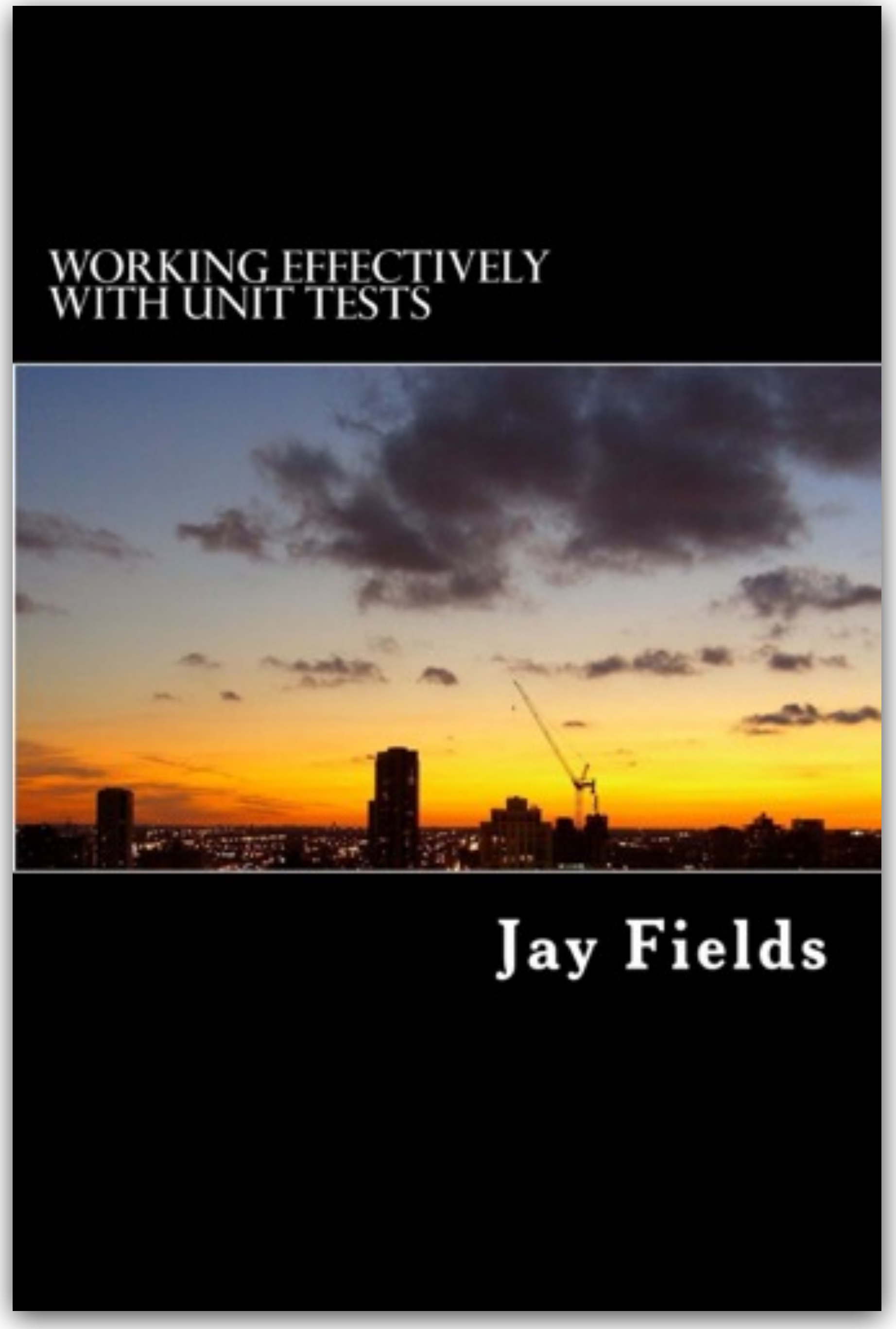


ROI

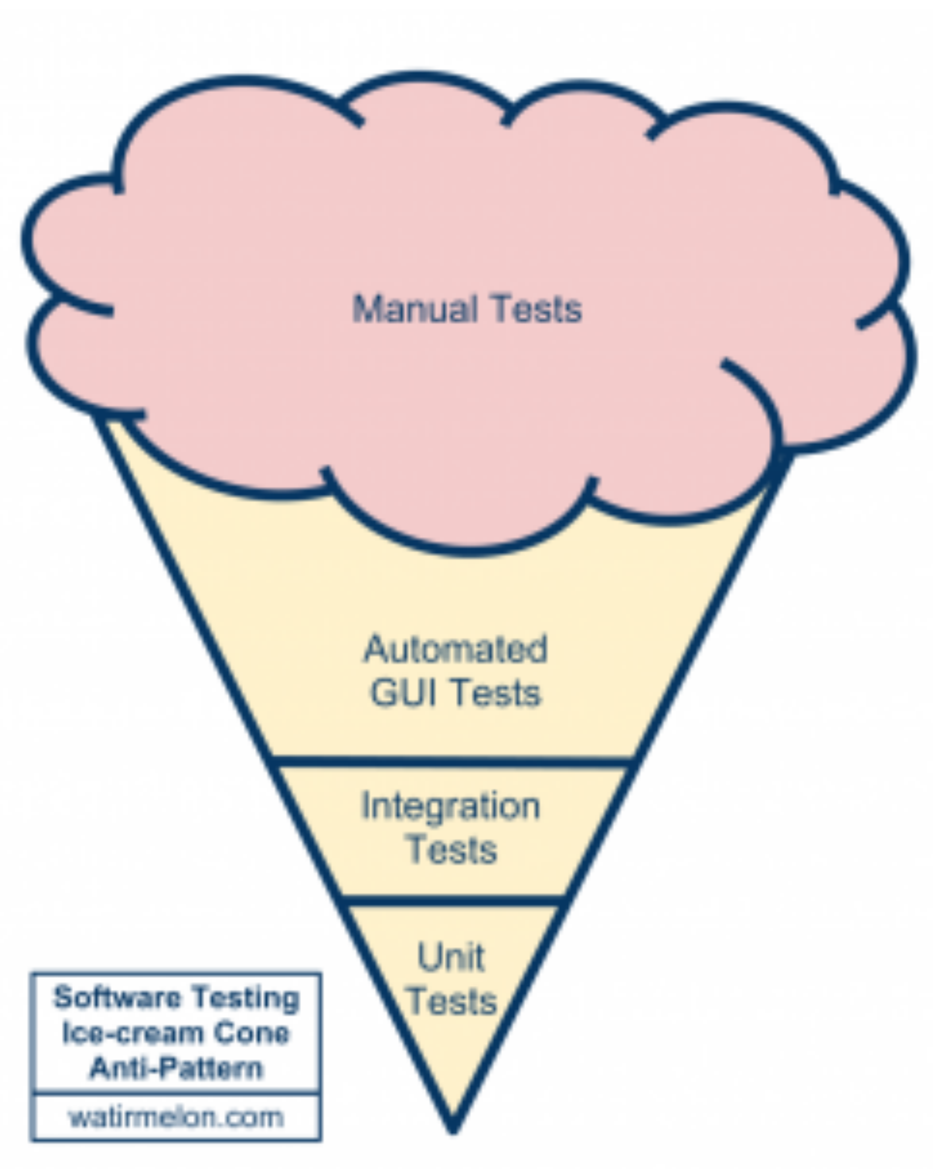
ROI

Return Of Investment

$$\frac{\text{Net profit}}{\text{Investment}}$$



Ice cream cone Anti Pattern



What Makes a Good Unit Test?

Repeatable/Deterministic

Isolated

Fast

Readable (Arrange-Act-Assert)

...

Writing tests is easy

Break
the
dependencies

Writing tests is easy

Direct input and output

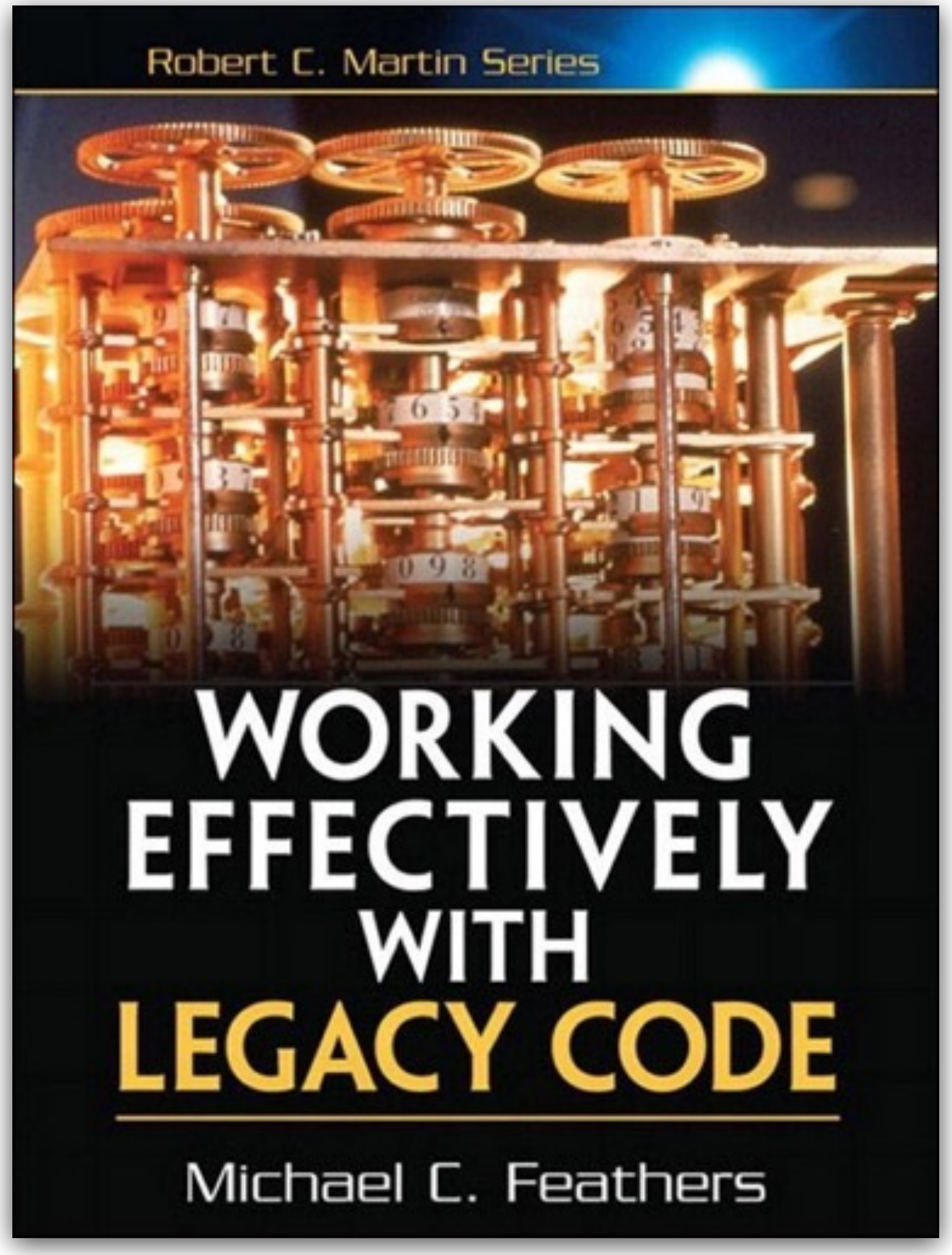
Writing tests is not easy :(

Direct input and output

Indirect input and/or output

Legacy code

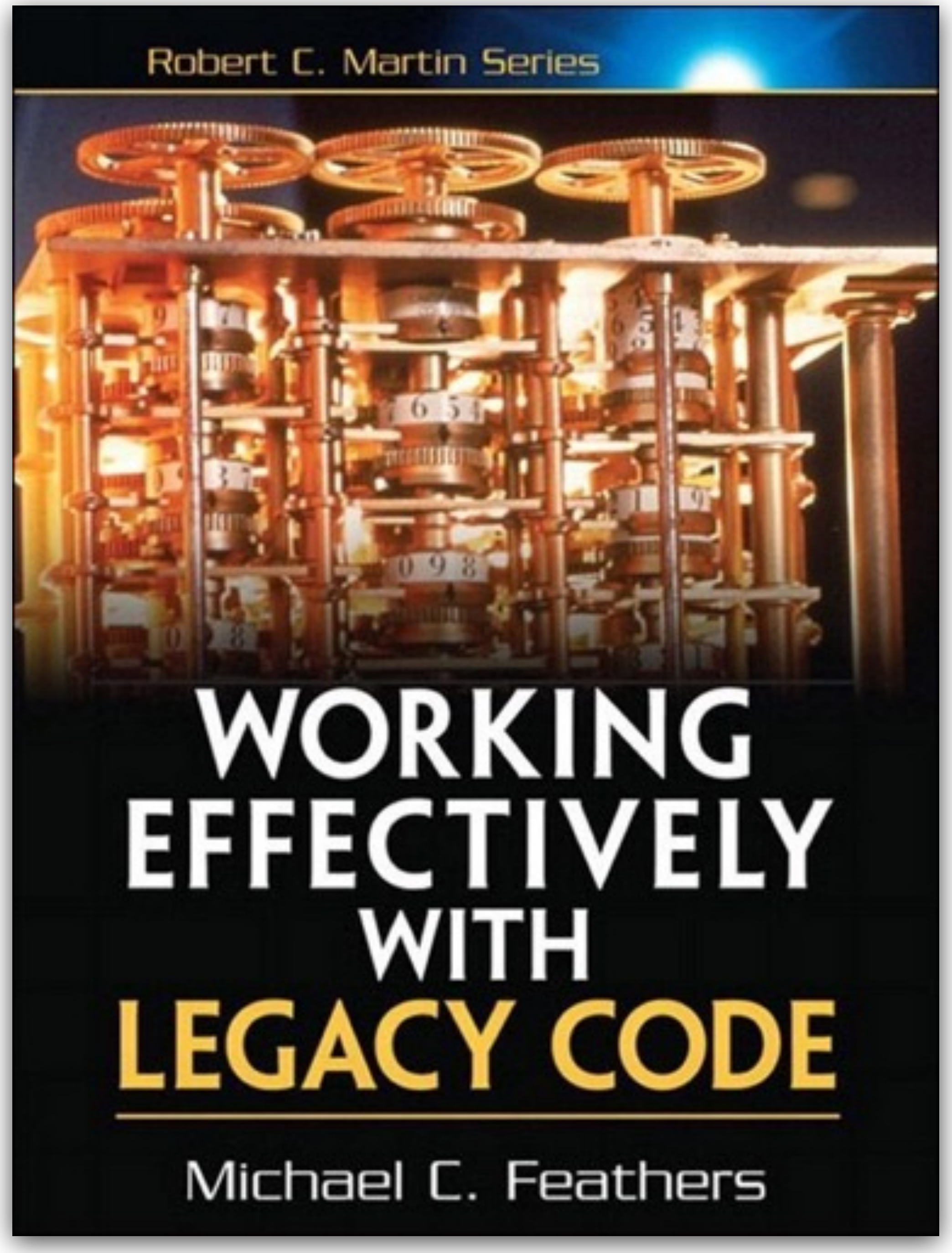
Legacy code


Robert C. Martin Series
WORKING EFFECTIVELY WITH LEGACY CODE
Michael C. Feathers

Legacy code

Legacy code is code without unit tests

Edit and pray
Vs
Cover and modify



Test After Development

Write the feature implementation

Do some manual testing

Try to write automatic tests

Modify the initial implementation to test it

“Standard” Android code is not testable :(

PostBatch

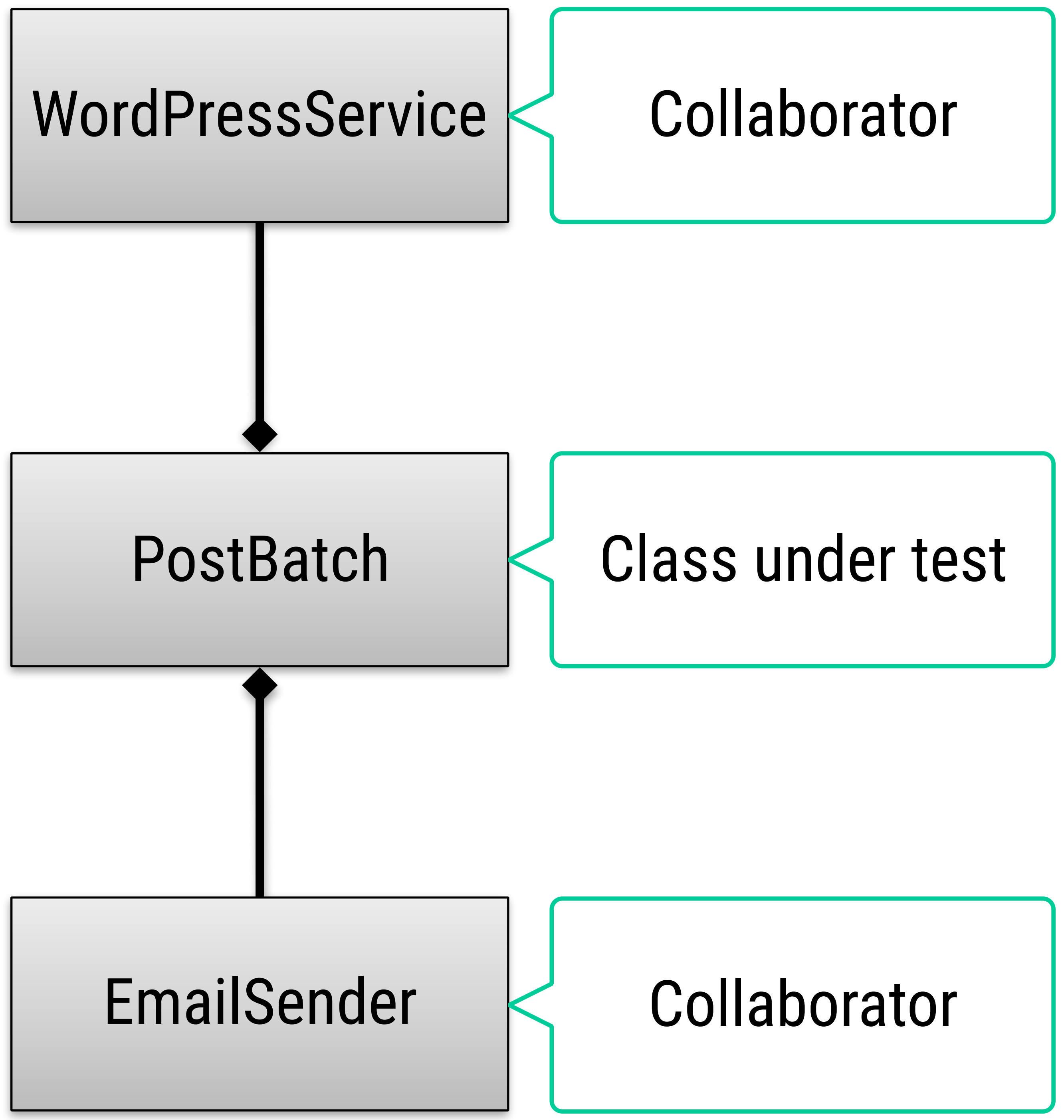
```java
public void execute() {
    PostResponse postResponse =
            WordPressService.getInstance().listPosts();

    EmailSender emailSender = new EmailSender();

    List<Post> posts = postResponse.getPosts();

    for (Post post : posts) {
        if (!post.isNotified()) {
            emailSender.sendEmail(post);
        }
    }
}
```

Legacy code dilemma

When we change code,
we should have tests in place.

To put tests in place,
we often have to change code.

Michael Feathers


WordPressService
Collaborator
PostBatch
Class under test
EmailSender
Collaborator

PostBatchTest

```java
public class PostBatchTest {

  private PostBatch postBatch = new PostBatch();

  @Test
  public void testExecute() {
    postBatch.execute();
    //???
  }
}
```

```java
public void execute() {
    PostResponse postResponse =
            WordPressService.getInstance().listPosts();

    EmailSender emailSender = new EmailSender();

    List<Post> posts = postResponse.getPosts();

    for (Post post : posts) {
        if (!post.isNotified()) {
            emailSender.sendEmail(post);
        }
    }
}
```

```java
public void execute() {
    PostResponse postResponse =
            WordPressService.getInstance().listPosts();
   EmailSender emailSender = new EmailSender();
    List<Post> posts = postResponse.getPosts();
    for (Post post : posts) {
        if (!post.isNotified()) {
            sendEmail(emailSender, post);
        }
    }
}

protected void sendEmail(EmailSender sender, Post post) {
    sender.sendEmail(post);
}
```

```java
public class PostBatchTest {
    private static final String MESSAGE = "abc";
    private Post sentPost;
    private PostBatch postBatch;

    @Before
    public void setUp() throws Exception {
        WordPressService.setInstance(new WordPressService() {
            @Override
            public PostResponse listPosts() {
                return new PostResponse(new Post(MESSAGE));
            }
        });
        postBatch = new PostBatch() {
            @Override
            protected void sendEmail(EmailSender sender, Post post) {
                sentPost = post;
            }
        };
    }

    @Test
    public void execute() throws Exception {
        postBatch.execute();
        assertThat(sentPost.getMessage()).isEqualTo(MESSAGE);
    }
}
```

Legacy code

Not the perfect solution

First step to increase coverage

Then modify and refactor

Dependency Injection

Inversion Of Control

```java
private WordPressService wordPressService;

private EmailSender emailSender;

public PostBatch(WordPressService wordPressService,
    EmailSender emailSender) {
  this.wordPressService = wordPressService;
  this.emailSender = emailSender;
}

public void execute() {
  PostResponse postResponse = wordPressService.listPosts();
  List<Post> posts = postResponse.getPosts();
  for (Post post : posts) {
    if (!post.isNotified()) {
      emailSender.sendEmail(post);
    }
  }
}
```

Dependency Injection

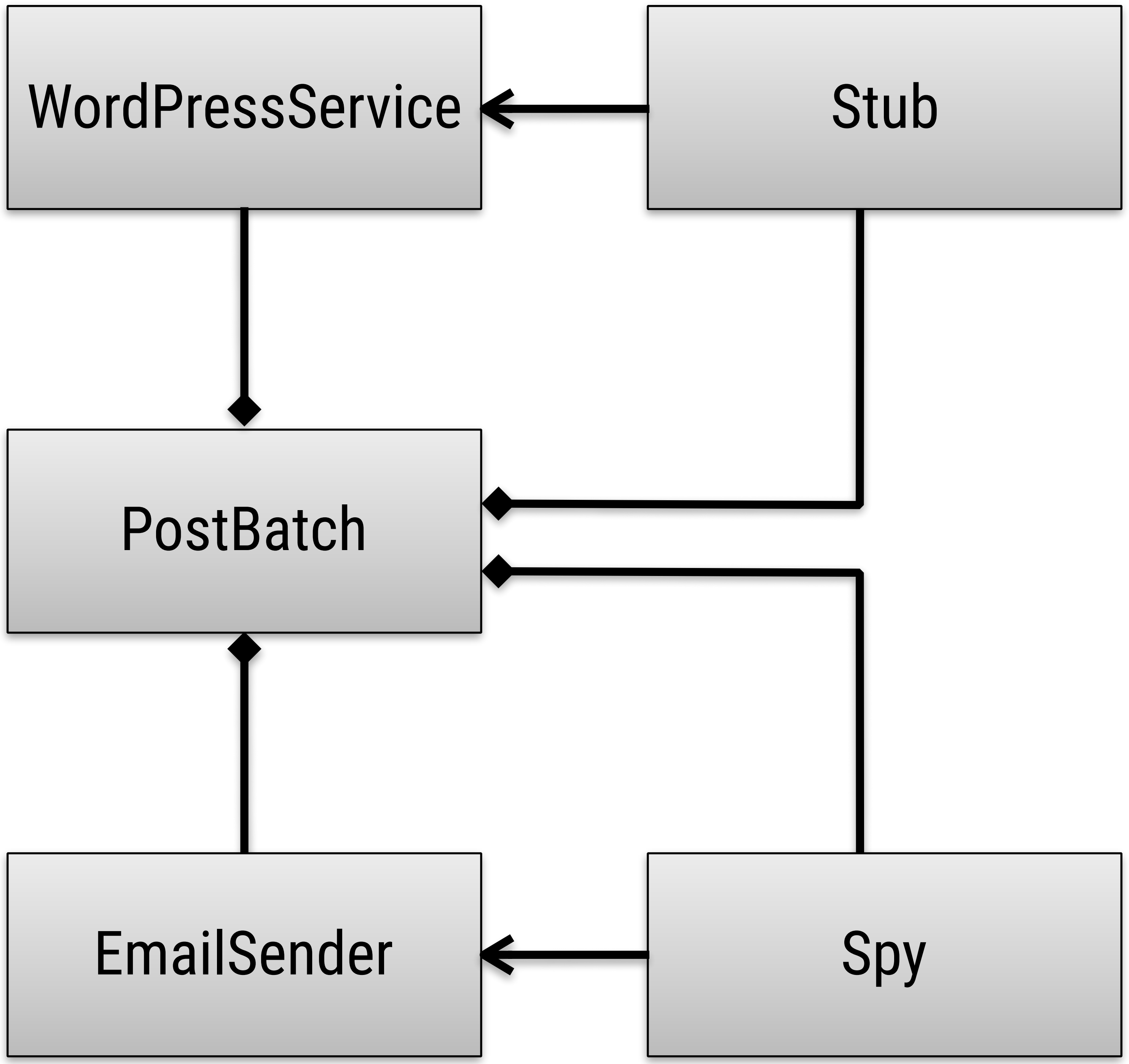
```java
public class Main {

  public static void main(String[] args) {
    new PostBatch(
      WordPressService.getInstance(), new EmailSender()
    ).execute();
  }

}
```

WordPressServiceStub

```java
public class WordPressServiceStub
    implements WordPressService {

  private PostResponse postResponse;

  public WordPressServiceStub(PostResponse postResponse) {
    this.postResponse = postResponse;
  }

  @Override public PostResponse listPosts() {
    return postResponse;
  }
}
```

EmailSenderSpy

```java
public class EmailSenderSpy extends EmailSender {

  private int emailCount;

  @Override public void sendEmail(Post p) {
    emailCount++;
  }

  public int getEmailCount() {
    return emailCount;
  }
}
```


WordPressService
Stub
PostBatch
EmailSender
Spy

Test doubles

```java
private PostBatch postBatch;

private EmailSenderSpy emailSenderSpy;

private WordPressServiceStub serviceStub;

@Before public void init() {
  emailSenderSpy = new EmailSenderSpy();
  serviceStub = new WordPressServiceStub(
    new PostResponse(new Post(), new Post(), new Post())
  );
  postBatch = new PostBatch(serviceStub, emailSenderSpy);
}

@Test
public void testExecute() {
  postBatch.execute();
  assertEquals(3, emailSenderSpy.getEmailCount());
}
```

Mockito

Mockito

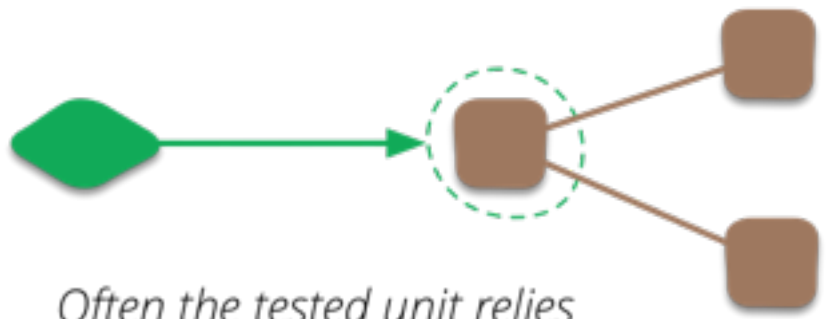
```java
private WordPressService service;
private EmailSender emailSender;
private PostBatch postBatch;

@Before public void init() {
  emailSender = Mockito.mock(EmailSender.class);
  service = Mockito.mock(WordPressService.class);
  postBatch = new PostBatch(service, emailSender);
}

@Test public void testExecute() {
  when(service.listPosts()).thenReturn(
    new PostResponse(new Post(), new Post(), new Post()));

  postBatch.execute();

  verify(emailSender, times(3)).sendEmail(any(Post.class));
}
```

Arrange Act Assert

@InjectMocks

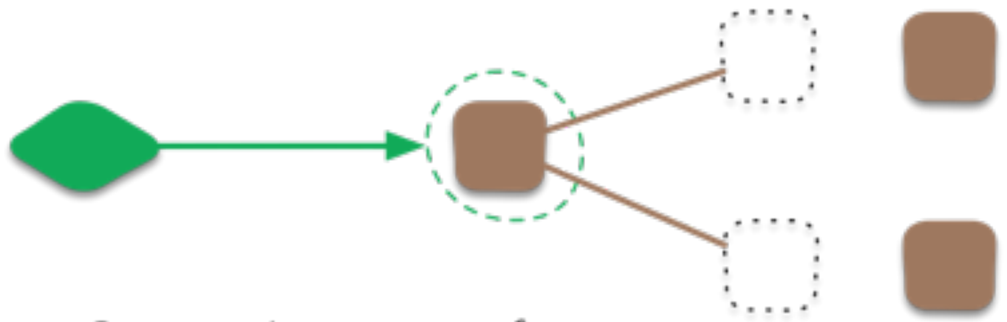
```java
public class PostBatchTest {

  @Rule MockitoRule rule = MockitoJUnit.rule();

  @Mock WordPressService service;

  @Mock EmailSender sender;

  @InjectMocks PostBatch postBatch;

  @Test
  public void testExecute() {
    when(service.listPosts()).thenReturn(
      new PostResponse(new Post(), new Post(), new Post()));

    postBatch.execute();

    verify(sender, times(3)).sendEmail(any(Post.class));
  }
}
```

Sociable Tests



Often the tested unit relies on other units to fulfill its behavior

Solitary Tests



Some unit testers prefer to isolate the tested unit