

Agile Software Development

Produced
by

Eamonn de Leastar (edelestar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



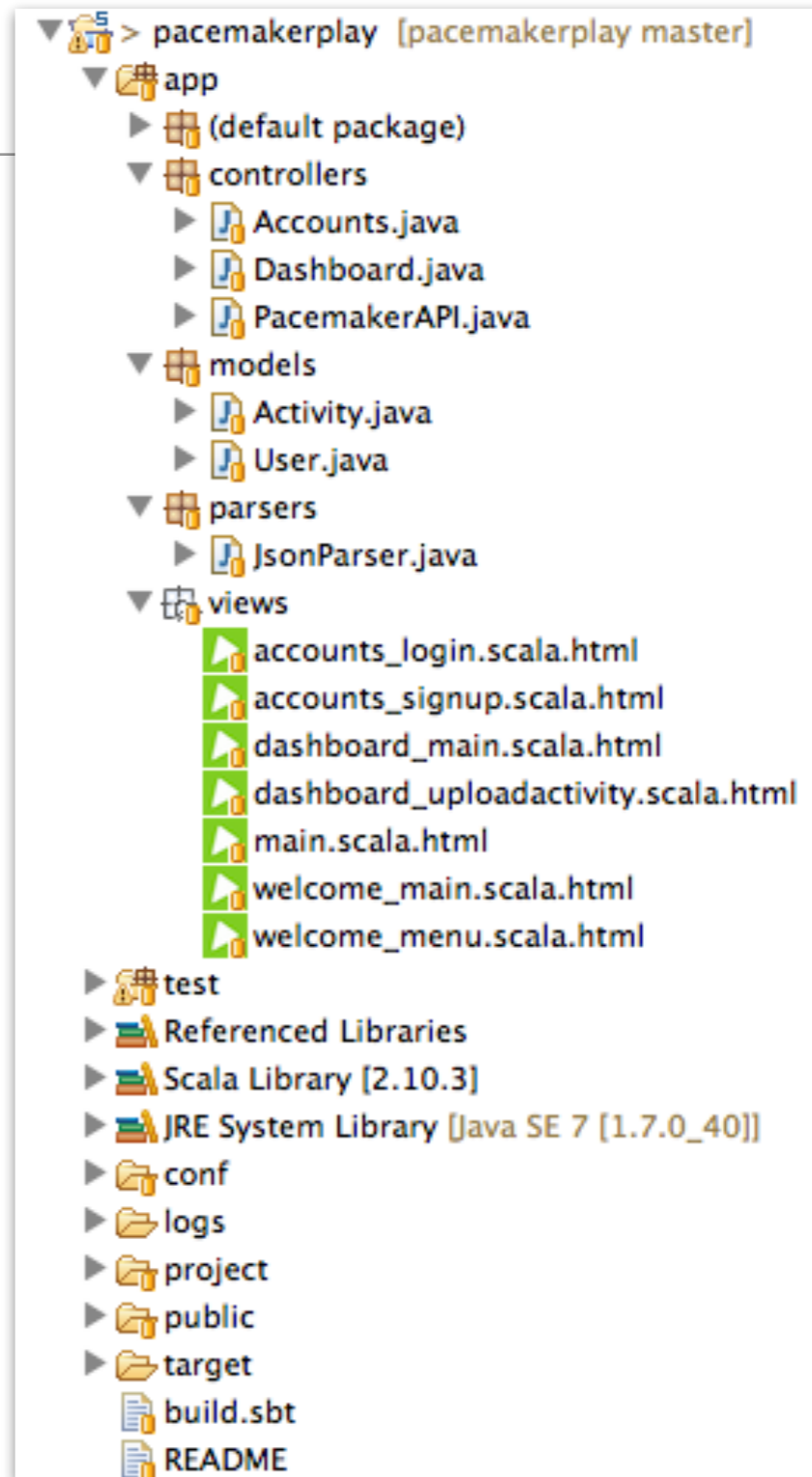
Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRCE



Mocking Opportunities in Pacemaker

Pacemaker 2.0

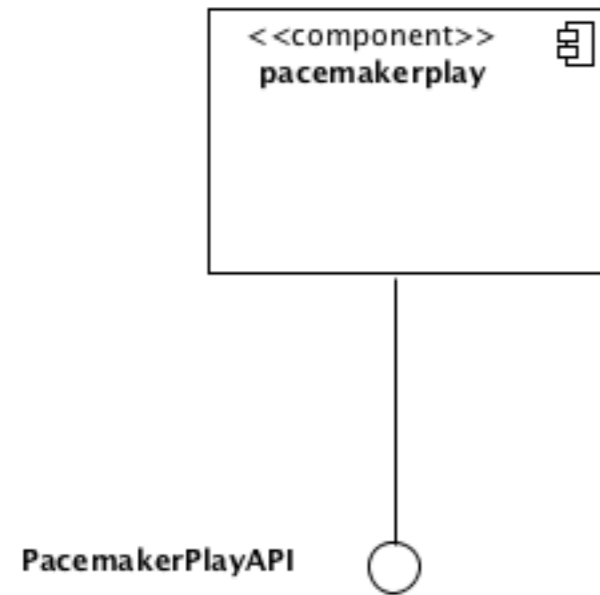
- REST Web Service
- Standard Web UI



Assignment Rubric for Assignment 2 (top marks deployment + any 2 others)

Standard	Deployment	Features	UX	DX
Baseline	REST (Local)	activities <i>(see runkeeper)</i>	Console	Rest Tests
Good	REST (Deployed)	reports <i>(see runkeeper)</i>	Console - asciiart	Models
Excellent	REST (Secured)	friends <i>(see runkeeper)</i>	Simple Web	Test Doubles (Factor out pacemaker into 2 services)
Outstanding	REST (2 x cloud)	dashboard <i>(see runkeeper)</i>	Web Ajax or App	API Documentation

pacemakerplay service



- Provides an API for managing:
 - users
 - activities
 - routes (within activities)

```
# API

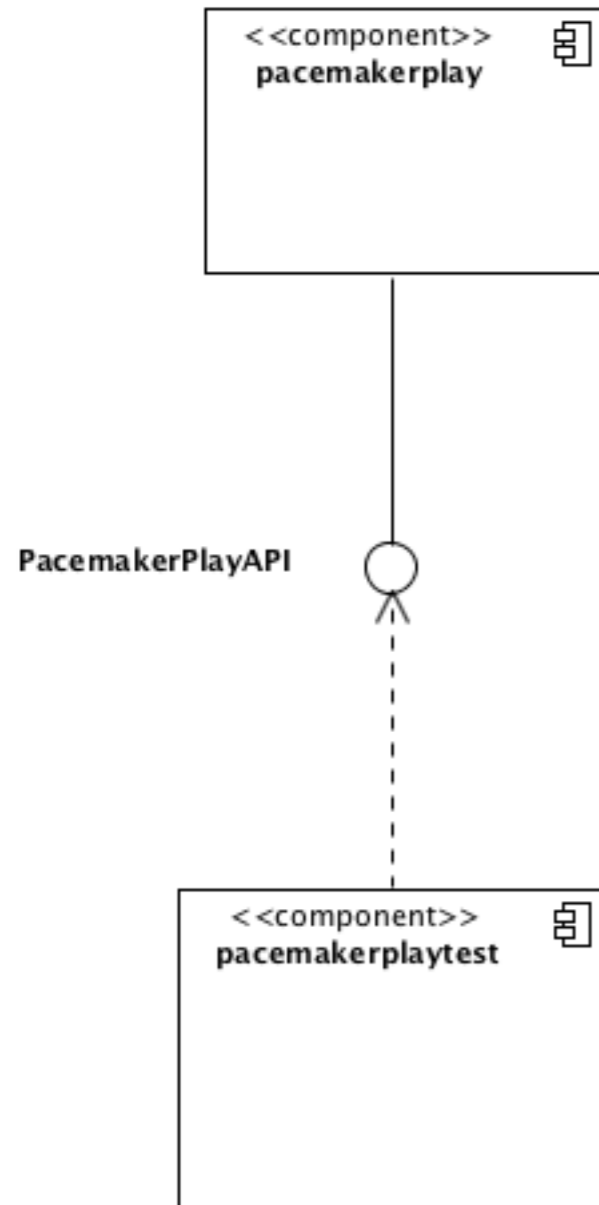
GET    /api/users           controllers.PacemakerAPI.users()
DELETE /api/users           controllers.PacemakerAPI.deleteAllUsers()
POST   /api/users           controllers.PacemakerAPI.createUser()

GET    /api/users/:id        controllers.PacemakerAPI.user(id: Long)
DELETE /api/users/:id        controllers.PacemakerAPI.deleteUser(id: Long)
PUT    /api/users/:id        controllers.PacemakerAPI.updateUser(id: Long)

GET    /api/users/:userId/activities controllers.PacemakerAPI.activities(userId: Long)
POST   /api/users/:userId/activities controllers.PacemakerAPI.createActivity(userId: Long)

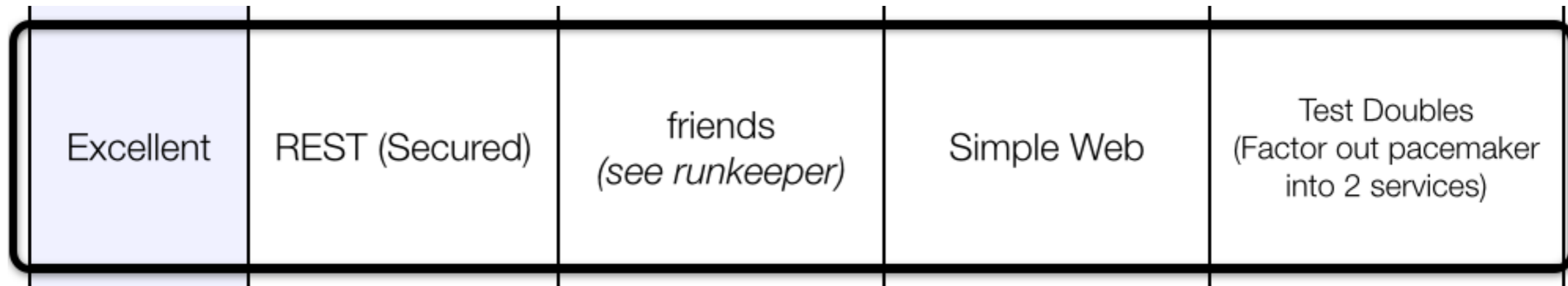
GET    /api/users/:userId/activities/:activityId controllers.PacemakerAPI.activity(userId: Long, activityId: Long)
DELETE /api/users/:userId/activities/:activityId controllers.PacemakerAPI.deleteActivity(userId: Long, activityId: Long)
PUT    /api/users/:userId/activities/:activityId controllers.PacemakerAPI.updateActivity(userId: Long, activityId: Long)
```

pacemakerplaytest



- Exercise the API over HTTP
- Full set of tests to verify key features

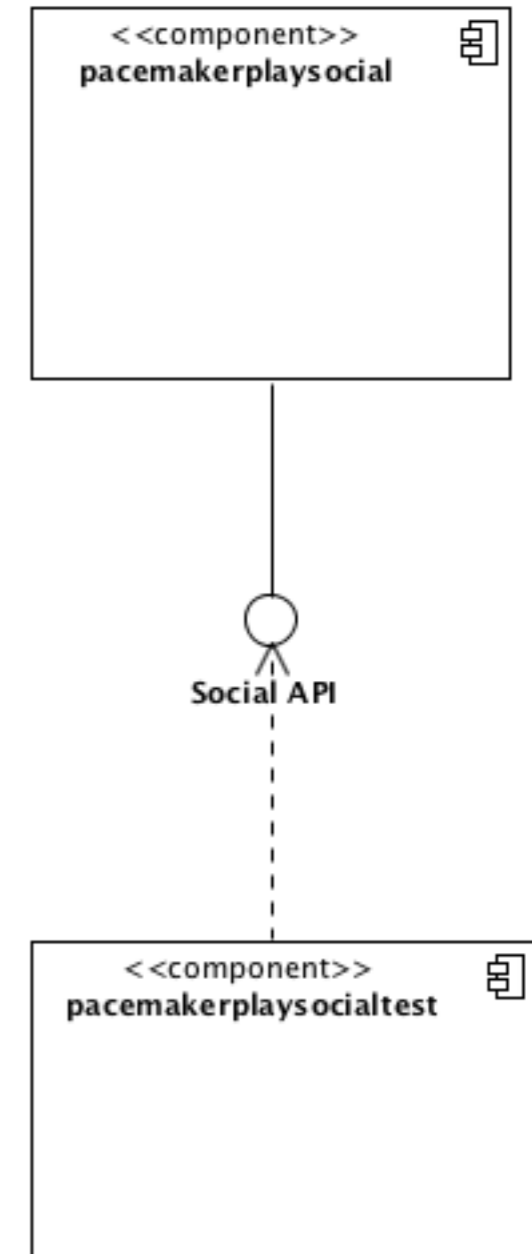
New Feature - 'Social'



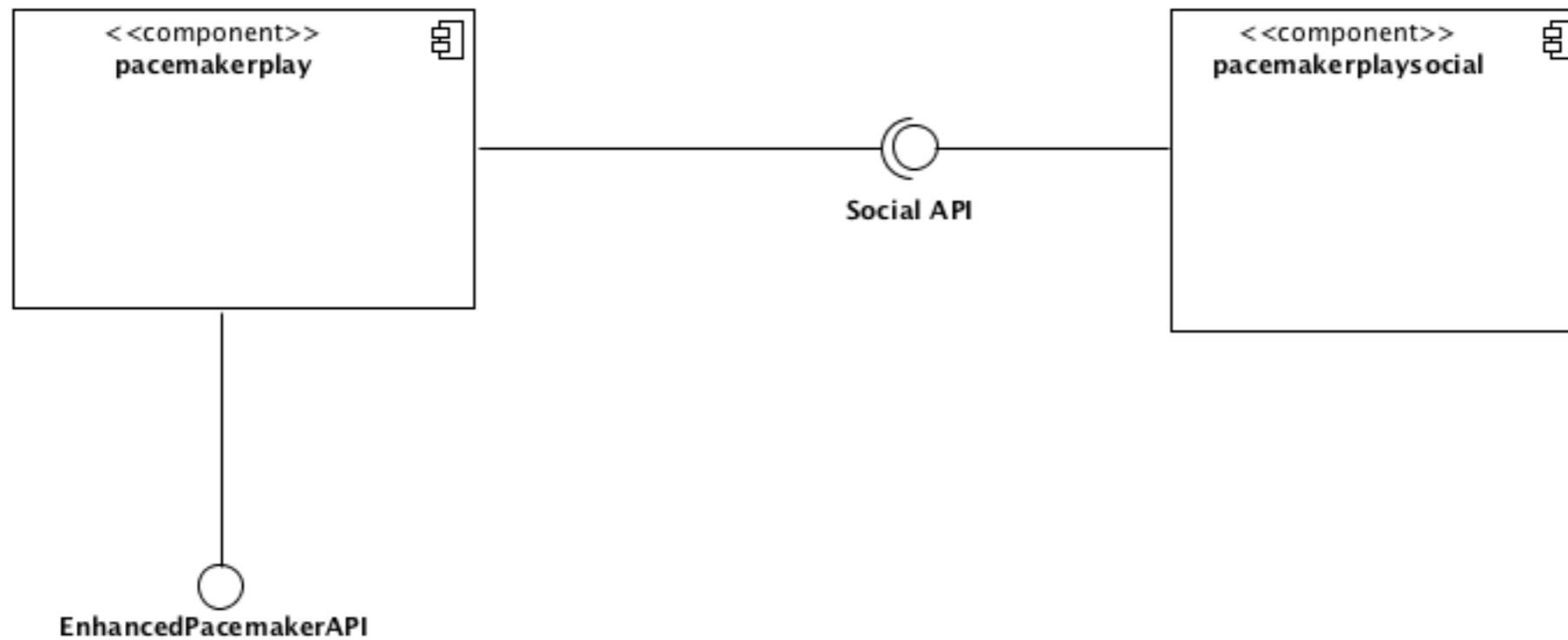
- Follow friends
- View Friends activities
- 'Feeds', etc...
- Consider modelling this as a separate service
- With its own API for managing
 - social graph
 - updates
 - follow/unfollow etc...

pacemakerplaysocial

- Uncouple the social aspects from the core activity service
- Allows the social service to be constructed and optimised independently
 - NoSQL database more appropriate
 - Interfaces to twitter/ facebook etc..



pacemakerplay -> pacemakerplaysocial

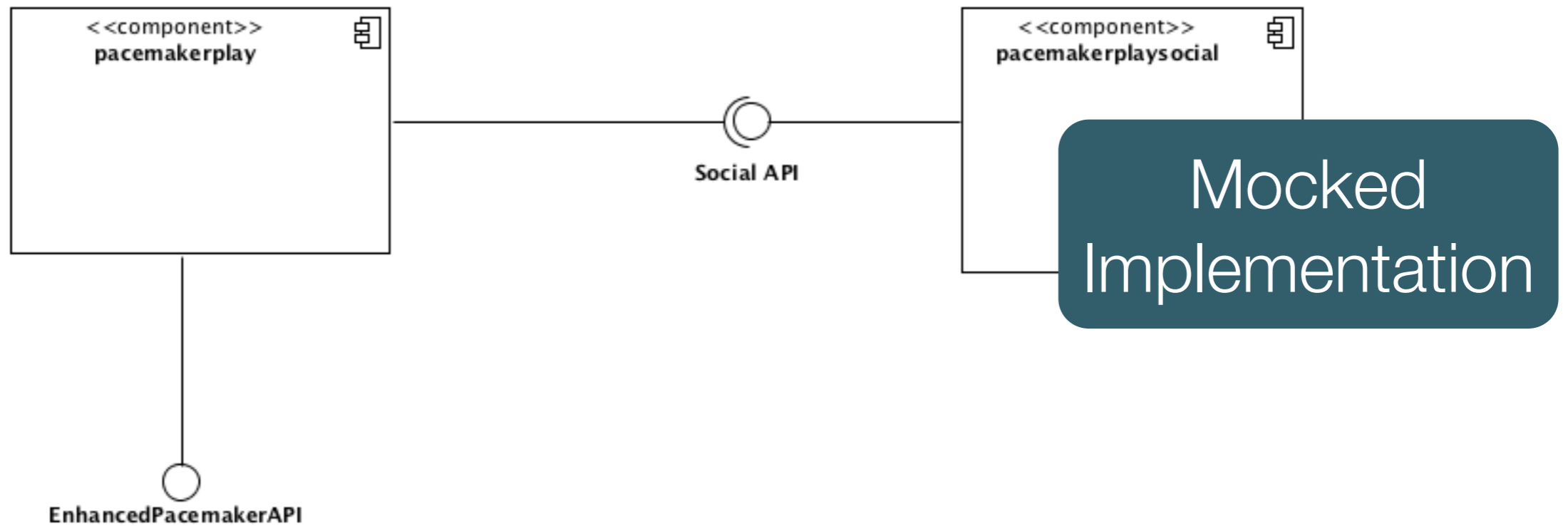


- ‘Enhanced’ API uses Social features, provided by pacemakerplaysocial service

The Role of Mock Objects

- Postpone the Construction of the pacemakerplaysocial component
- Model the API first as a REST API
- Mock out its implementation
- Then:
 - Write Tests against the Mocked out API
 - Enhance pacemaker play API to incorporate social features

pacemakerplay -> pacemakerplaysocial



- Use REST Mocking Services to deliver realistic test data to pacemakerplay

- [Getting Started](#)
- [Stubbing](#)
- [Verifying](#)
- [Proxying](#)
- [Record and Playback](#)
- [Stateful Behaviour](#)
- [Simulating Faults](#)

WireMock

WireMock is a flexible library for stubbing and mocking web services. Unlike general purpose mocking tools it works by creating an actual HTTP server that your code under test can connect to as it would a real web service.

It supports HTTP response stubbing, request verification, proxy/intercept, record/playback of stubs and fault injection, and can be used from within a unit test or deployed into a test environment.

Although it's written in Java, there's also a JSON API so you can use it with pretty much any language out there.

What's it for?

Some scenarios you might want to consider WireMock for:

- Testing mobile apps that depend on third-party REST APIs
- Creating quick prototypes of your APIs
- Injecting otherwise hard-to-create errors in 3rd party services
- Any unit testing of code that depends on a web service

Mocky



Tweet 231

Mock your HTTP responses to test your REST API

```
> PUT http://www.mocky.io/v2/5185415ba171ea3a00704eed
```

```
< HTTP/1.1 200 OK  
< Content-Type: application/json; charset=UTF-8  
{ "hello": "world" }
```

Now with **JSONP** support, just add **?callback=myfunction** to your links.

Generate your custom response

Status Code

Content Type

Custom headers :

Eg: ETag, If-None-Match, Expires, Last-Modified, Server, X-Cache, Cache-Control, X-Frame-Options, Server, Set-Cookie, X-UA-Compatible...

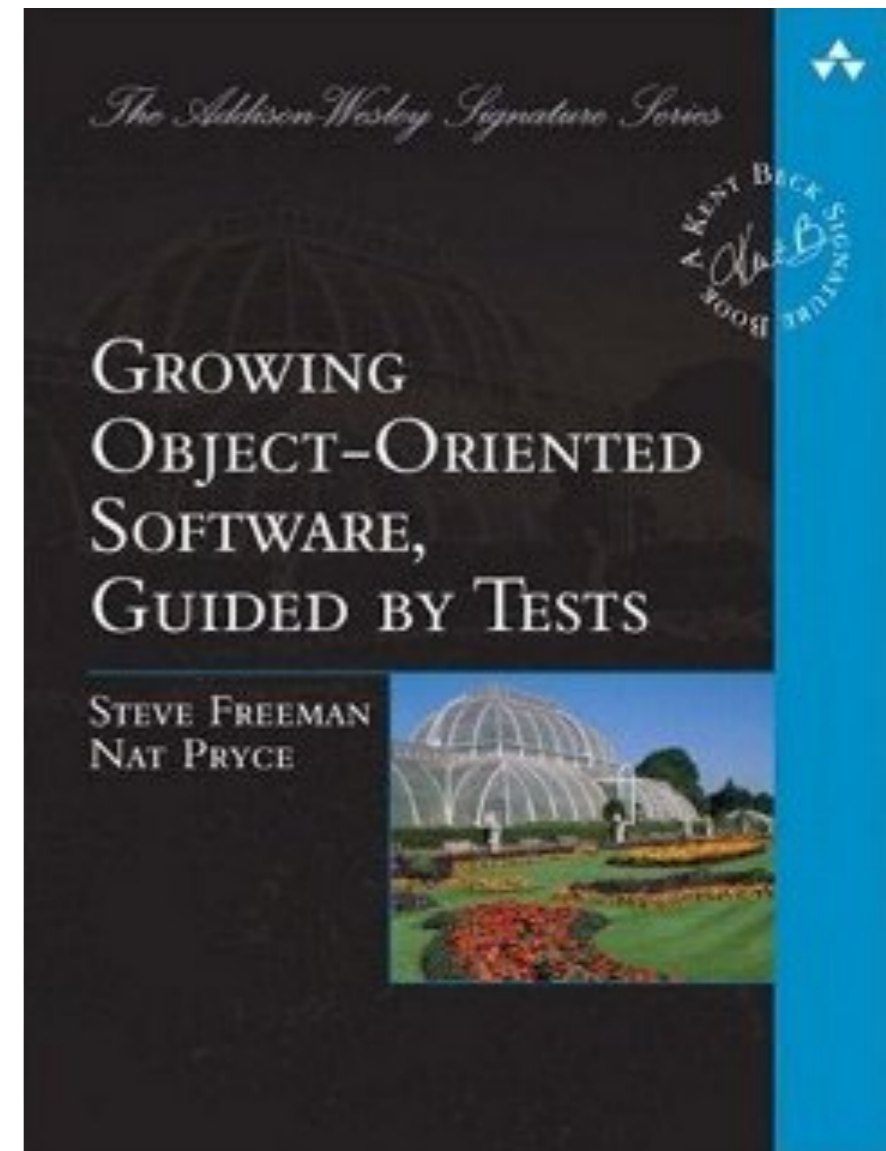
Body

[Generate my HTTP Response](#)

[Switch to basic mode](#)

Key Text on Employing Test Doubles Effectively

- Implementing TDD effectively: getting started, and maintaining your momentum throughout the project
- Creating cleaner, more expressive, more sustainable code
- Using tests to stay relentlessly focused on sustaining quality
- Understanding how TDD, Mock Objects, and Object-Oriented Design come together in the context of a real software development project
- Using Mock Objects to guide object-oriented designs
- Succeeding where TDD is difficult: managing complex test data, and testing persistence and concurrency



TDD - Key Texts

The Pragmatic Programmers

Pragmatic Unit Testing

In Java with JUnit
The Pragmatic Starter Kit - Volume II



Andrew Hunt David Thomas

The Addison-Wesley Signature Series

xUNIT TEST PATTERNS

REFACTORING
TEST CODE

GERARD MESZAROS



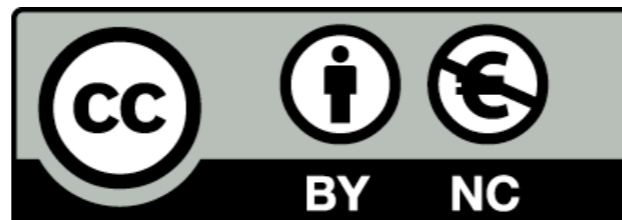
Foreword by Martin Fowler

The Addison-Wesley Signature Series

GROWING OBJECT-ORIENTED SOFTWARE, GUIDED BY TESTS

STEVE FREEMAN
NAT PRYCE





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

