

# Win at Life

with Unit Testing.

by Mark Story



# Who is this goofball

- Art college graduate that needed to make money.
- CakePHP core contributor for 2.5 years. Developer of DebugKit, ApiGenerator and several other plugins.
- Lead front end developer at FreshBooks



Can unit testing  
make me  
awesome?



Hell yes.



# Why write tests?

- Helps ensure things that did work, still do.
- Helps reduce time it takes to fix new defects.
- Helps you plan a design, showing you where it might be complicated or overly coupled.
- Living documentation.





- Catch issues earlier.
- Increased developer confidence.
- Automated killer robots.



# What to test?

- Code that keeps you up at night.
- Code that involves real money.
- Code that has broken before.
- Code that is tedious to test manually.



# Different flava's

- Unit tests
- Functional or Integration tests



# Unit Tests

- Testing the smallest parts of your application in isolation.
- Often testing single methods or single functions.
- Generally uses many mocks.
- Really useful when doing Test driven development.



# Functional Tests

- Tests pieces working well together.
- Test top level objects to ensure all their guts are working together.
- Tests hitting database fixtures are often functional tests.
- Functional tests run slower, but use more active code.



# Challenges of testing

- Time. It takes time to write tests. But it takes more time to fix the same bugs later multiple times.
- Only shows the presence of issues, not the absence.
- Only catches errors you have tests for.



# Benefits of testing

- Find problems earlier.
- Can be automated, robots are helpful.
- Less manual testing.
- Easier to ensure requirements are met.
- Unicorns.



# The End.

Of the part where I extol the winsauce tests  
can bring.



# Mock objects

<insert>A funny joke</insert>



# Mocks and test doubles

- Writing isolated tests is hard. Mock objects make it easier.
- Mocks help ensure your objects are touching other objects in the right places.
- Mocks help you stub out behaviour before you write the real deal.



Can I use a mock  
for:

Global functions?

Stuff not in functions?



No.



# When to use mocks

- When talking to the outside world.
  - Email.
  - Webservices (Twitter, Paypal, YouTube).
  - Writing files to disk.
  - Writing to the database.





- When you want to ensure two objects are working together properly.
- When you do want to isolate problems.



# Mocking your code

- Make sure dependencies can be replaced at runtime.
  - Constructor injection.
  - Factory methods.
  - Setter methods.



# Constructor Injection

- Accept objects your object depends on in the constructor.
- Jam mocks in when testing.

```
<?php
class Car {
    function __construct($engine, $driver) {
        $this->_engine = $engine;
        $this->_driver = $driver;
    }
}
```



# Factory Factory Factory

- Factory methods can be overridden in subclasses to return mocks.

```
<?php
class RaceCar {
    function __construct($driver) {
        $this->_engine = $this->_getEngine();
        $this->_driver = $driver;
    }

    function _getEngine() {
        if (empty($this->_engine)) {
            $this->_engine = new BigEngine();
        }
        return $this->_engine();
    }
}
```



# Setter method

- Allows post construction modification of internal dependencies.

```
<?php
class MonsterTrunk {
    ▶ function __construct($driver) {
    ▶     ▶ $this->_engine = new GiantEngine();
    ▶     ▶ $this->_driver = $driver;
    ▶ }
    ▶
    ▶ function setEngine($engine) {
    ▶     ▶ $this->_engine = $engine;
    ▶ }
}
}
```



# Mocks in the house

- Mocks can be critical of a test subject.
- Mocks and stub out dangerous components, or methods.



# Critical mocking

- Critic mocks check that other objects touch them correctly.

```
<?php  
$RequestHandler->response = $this->getMock('CakeResponse', array('type'));  
$RequestHandler->response->expects($this->at(0))  
    ->method('type')  
    ->with('application/json');  
$RequestHandler->response->expects($this->at(1))  
    ->method('type')  
    ->with('text/xml');  
}  
}
```



# Stub danger.

- You can stub methods that send headers or touch the outside world.

```
$Controller->response = $this->getMock(
    'CakeResponse', array('header', 'statusCode')
);
$Controller->response->expects($this->once())
    ->method('statusCode')
    ->with(301);
$Controller->response->expects($this->once())
    ->method('header')
    ->with('Location', 'http://cakephp.org');
```



# Stub expensive stuff.

- Mocks can be used to stub network resources or expensive to fetch results.

```
$contactData = array(
    array('name' => 'Jimbo Jenkins', 'email' => 'j.jenkins@example.com'),
    ...
);
$Service = $this->getMock('CampaignMonitorService');
$Service->expects($this->once())
    ->method('getContacts')
    ->will($this->returnValue($contactData));
```



# In conclusion

- Mocks help make tests run faster, and require less setup & teardown.
- They allow you to write *true* unit tests.



The End.

(of the part where I blabber about mocks)



# Automated Killer Robots

test automation for squishy humans.



# Automated testing.

- Tests that are never run, are as good as deleted.
- Build servers to the rescue!



# Hudson

- Hudson is a continuous integration server built in Java.
- Has a great plugin community.
- Can send emails, Jabber and give pithy Chuck Norris quotes.





- Can be configured to run on a schedule or post commit.
- Integrates well with SVN and there is a GIT plugin too.



# Typical Hudson setup

- Run tests on each commit/push.
- Run all tests again each night, incase there was a slow day.
- Nightly builds are also a good time to run code coverage builds.



# Setting Hudson up

- Go to <http://hudson-ci.org>
- `wget http://hudson-ci.org/latest/hudson.war`
- `java -jar hudson.war`



# Hudson dashboard

The screenshot shows the Hudson dashboard with a blue header bar containing the 'Hudson' logo, a search box, and a help icon. Below the header, there are navigation links for 'New Job', 'Manage Hudson', 'People', and 'Build History'. A green arrow points to the 'New Job' link. On the right, there are links for 'ENABLE AUTO REFRESH' and 'add description'. The main content area features a table with columns for 'S', 'W', 'Job', 'Last Success', 'Last Failure', and 'Last Duration'. A single row is visible for a job named 'Test build.'. Below the table, there are links for 'Icon: S M L' and a legend for RSS feeds. On the left, there are sections for 'Build Queue' (showing no builds) and 'Build Executor Status' (showing two idle executors). A faint watermark of a man's face is visible in the bottom left corner.

## Hudson

search ?

[Hudson](#) [ENABLE AUTO REFRESH](#)

[New Job](#) [Manage Hudson](#) [People](#) [Build History](#) [add description](#)

S	W	Job ↓	Last Success	Last Failure	Last Duration
		<a href="#">Test build.</a>	6 min 7 sec (#7)	7 min 19 sec (#6)	0.78 sec

Icon: [S](#) [M](#) [L](#)

[Legend](#) for all for failures for just latest builds

### Build Queue

No builds in the queue.

### Build Executor Status

#	Status
1	Idle
2	Idle



# Hudson dashboard

**Hudson** search ?

[Hudson](#) ENABLE AUTO REFRESH

[add description](#)

[New Job](#)

[Manage Hudson](#)

[People](#)

[Build History](#)

**Build Queue**

No builds in the queue.

**Build Executor Status**

#	Status
1	Idle
2	Idle

S	W	Job ↓	Last Success	Last Failure	Last Duration
		<a href="#">Test build.</a>	6 min 7 sec (#7)	7 min 19 sec (#6)	0.78 sec

Icon: [S](#) [M](#) [L](#)

[Legend](#) for all for failures for just latest builds



# Hudson dashboard

The screenshot shows the Hudson dashboard with a blue header bar containing the 'Hudson' logo, a search box, and a help icon. Below the header, there are navigation links for 'New Job', 'Manage Hudson', 'People', and 'Build History'. On the right, there are links for 'ENABLE AUTO REFRESH' and 'add description'. The main content area features a table of build jobs. The table has columns for 'S' (Success icon), 'W' (Warning icon), 'Job', 'Last Success', 'Last Failure', and 'Last Duration'. A single job is listed: 'Test build.' with a success icon, a warning icon, a duration of '6 min 7 sec (#7)', and a failure duration of '7 min 19 sec (#6)'. Below the table, there are links for 'Icon: S M L' and three RSS feed links: 'Legend', 'for all', 'for failures', and 'for just latest builds'. On the left side, there are two summary boxes: 'Build Queue' (No builds in the queue) and 'Build Executor Status' (a table with 2 idle executors). A faint watermark of a man's face is visible in the bottom left corner.

## Hudson

search ?

[Hudson](#) [ENABLE AUTO REFRESH](#)

[New Job](#) [add description](#)

[Manage Hudson](#)

[People](#)

[Build History](#)

S	W	Job ↓	Last Success	Last Failure	Last Duration
		<a href="#">Test build.</a>	6 min 7 sec (#7)	7 min 19 sec (#6)	0.78 sec

Icon: [S](#) [M](#) [L](#)

[Legend](#) [for all](#) [for failures](#) [for just latest builds](#)

### Build Queue

No builds in the queue.

### Build Executor Status

#	Status
1	Idle
2	Idle



# Hudson dashboard

**Hudson**  ?

[Hudson](#) [ENABLE AUTO REFRESH](#)

[New Job](#) [add description](#)

[Manage Hudson](#)

[People](#)

[Build History](#)

S	W	Job ↓	Last Success	Last Failure	Last Duration
		<a href="#">Test build.</a>	6 min 7 sec (#7)	7 min 19 sec (#6)	0.78 sec

Icon: [S](#) [M](#) [L](#)

[Legend](#) for all for failures for just latest builds

**Build Queue**  
No builds in the queue.

**Build Executor Status**

#	Status
1	Idle
2	Idle



# Hudson project

## Hudson

 ?

Hudson » Test build. ENABLE AUTO REFRESH

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Build Now](#)

[Delete Project](#)

[Configure](#)

### Project Test build.




Test project build.



[Workspace](#)

[Recent Changes](#)

[edit description](#)

#### Build History [\(trend\)](#)

	#7	<a href="#">30-Aug-2010 11:11:50 PM</a>
	#6	<a href="#">30-Aug-2010 11:10:37 PM</a>
	#5	<a href="#">30-Aug-2010 11:08:52 PM</a>

 for all  for failures

#### Permalinks

- [Last build \(#7\), 16 min ago](#)
- [Last stable build \(#7\), 16 min ago](#)
- [Last successful build \(#7\), 16 min ago](#)
- [Last failed build \(#6\), 17 min ago](#)
- [Last unsuccessful build \(#6\), 17 min ago](#)



The End.

For real this time.



Questions?