



Testing software with automated tests

Jonathan Haddock
codeHarbour - 19th May 2022

 <https://blog.jonsdocs.org.uk>
 @joncojonathan

I work in cybersecurity for my day job, but in my spare time I'm a developer. Please note that I'm not an expert on this topic, I'm just recounting some of my experiences so far.

There's been talks before - see the codeHarbour archive for talks by:
Glenn Wilson (<https://www.youtube.com/watch?v=rkGaj-NA678>)
Nick Ellis (<https://www.youtube.com/watch?v=wLFGprYlvCE>)

Tests don't add to the functionality of the product, so why bother? In this presentation we'll cover some of the reasons why it's in the developer's interest to perform tests.

Why test?

- Makes sure your code does what you intend
 - Functional tests
- Checks you've built the right thing
 - Acceptance tests
- Confirms if your change has just broken something
 - It happens!
- Helps find security issues
 - I speak from experience...🙈

The problem

- Testing software by hand takes time...
 - Is this option on screen?
 - What happens if I click this?...
- ...and doesn't scale
 - After every change!

Initially I would test all of my application by hand. This wasn't too bad at the beginning, just a few pages and operations, but as projects grow that simply doesn't scale. For user facing elements, getting a human to test is still a good idea at the acceptance testing stage (e.g. sign off by your client). The client might want some tweaks to the look of the system. Up until that point though automated tests can be incredibly beneficial.

Given that any change can break something, even something unrelated, it's worth running your tests again before deployment.

Automated tests - a solution

- There are many testing types
 - Static analysis
 - Dynamic analysis
 - Functional, acceptance
 - ...
- Also many testing tools
 - PHPStan (static analysis)
 - Codeception (dynamic analysis)
 - JUnit
 - ...

Glenn did a good talk on testing types, so I won't cover this in detail here. Do go check out his talk in the Code Harbour archive.

For this talk, I'm not interested in checks that your code meets a desired style. Your Integrated Development Environment (IDE) can almost certainly do that for you though.

Functional checks the code does what's expected.
Acceptance checks the right product was built.

I'm focusing on functional.

Some testing frameworks



Logos copyright their respective authors.

There's probably a test framework for your programming language of choice, it's just a case of learning how to use it. As we'll see during this talk, it's worth investing some time to learn.

I'm using PHP: Codeception, pcov and Yii2

- Most of my development is in PHP
- I've used Yii for a long time
- Codeception supports Yii2
- `pcov` is a code coverage driver
- The PHPStorm IDE integrates with Codeception, so tests can be easily run from the IDE if desired

When do you write the tests?

- After your code?
 - Are you then testing what you know you've developed?
- Before you code?
 - Test driven development
 - More likely to test required outcomes
 - An initial test run should fail - no code yet
 - If it passes, something is wrong

Personally I do a bit of both - a rebel!

There's a risk that if you write your tests after the code that you'll run out of time and never write the tests. This is particularly true if you're following a waterfall development model, which generally places testing towards the end of a project. It's also possible to introduce bias in your tests: "I know the system works this way, so if I write my test like this it will pass". You may not intend to write a biased test, but it could happen.

Writing your tests before your code requires some form of specification to test against. Once the test is written the test suite should be run and your new tests *should* fail because there's no code written for that feature yet. If the test passes then either you've written a bad test, or someone has already written the code (maybe a colleague, maybe you and you just forgot). Once the code is written, running the tests again should pass.

An example test

```
public function preventAnonymousCreationOfTag( FunctionalTester $I )
{
    // We do not login to ensure we are anonymous
    $I->enableSetting( settingKey: 'allowLocalLogin' );
    $I->amOnPage( page: '/tag/create' );
    $I->dontSee( text: 'Create Tag' );
    $I->see( text: 'eVitabu password' );
}
```

Naming things is hard! The test “prevent anonymous creating of tag” probably be renamed to “confirm anonymous users cannot create tags”.

First this test users a helper function (enableSetting) to enable a setting - using a helper function saves duplicate code.

amOnPage directs the test tool to go to the page for creating tags.

The test tool checks it cannot see the text “Create Tag”, which is present at the top of the tag creation form.

Finally the tool checks that “eVitabu password” can be seen, as that’s from the login screen.

Absolutely this test could be expanded to be more specific (checking there’s no tag creation form, confirming the password field is on the login page), but for my immediate purposes this is sufficient.

HTML report output

TagCest » Prevent anonymous creation of tag 0.02s

+ I enable setting "allowLocalLogin"

I am on page "/tag/create"

I don't see "Create Tag"

Failed asserting that on page /tag/create

--> Create Tag eVitabu Login Home Tags Create Tag Create Tag Name Keywords Create © African Pastors Fellowship 2022 | Privacy policy | Terms of service | Security APF Website

--> does not contain "Create Tag".

See [HTML snapshot](#) of a failed page

Codeception can output the results to your terminal, showing a count of tests that passed and failed or the step that failed. Personally I like to use the HTML reports that it produces. This shows the result of a failed test (the one we just saw).

The first two steps (enabling a setting and moving to the tag creation page) passed. Step three - "I don't see *create tag*" - failed, as that text was visible on the page.

Clicking the link to the HTML snapshot lets us see what was on showing in the browser at that time.

Looking at the failed test

[eVitabu](#) 

- [Login](#)

1. [Home](#)

2. [Tags](#)

3. Create Tag

Create Tag

Name

Keywords

© African Pastors Fellowship 2022 | [Privacy policy](#) | [Terms of service](#) | [Security](#)

[APF Website](#)

Note there's no CSS or Javascript included in the snapshot, just a skeleton view. From here we've got a lot more information to see what actually happened.

Identifying security concerns

That example test looked to check if anonymous access to an admin function was possible...

... it was.

Had the test not been present, that authentication bypass would have gone unnoticed!

As developers it's our job to write secure code to protect both the interests of the organisation and the individuals whose data we're storing. While automated tests are not a substitute for a web application penetration test, they are a good way to identify and fix problems before independent testers are consulted. This saves money 😊. Automated tests demonstrate mature development practices, and including security focused tests can help reassure auditors and investors that things are being done properly.

The previous slide showed a failed test for checking anonymous users couldn't create tags. I actually broke the authentication item of the application to get that result, as it had previously been fixed. After reverting my intentional sabotage I ran my tests again to make 100% certain I had not introduced a vulnerability.

Running tests, in any order

- Configurable
- Each run of the test suite runs them in a random order
- Confirms tests don't depend on one another

Codeception can be configured to run all database operations within transactions, meaning any changes can be undone once the tests complete. This does require a compatible database, for example MySQL or MariaDB.

If your tests have to run in a particular order to be successful then your tests depend on one another. This is a problem, particularly as end users may not perform their actions in the same order. Configure your test framework to run tests in a random order to help identify this problem (and to get better tests as a result). Obviously if you're running tests in a transaction, and database operations would be undone after each individual test which will help identify this too.

Code coverage

- The amount of your code tested by your, er, tests
- Don't strive for 100%
 - Potentially unattainable
 - Tests are good, but the client pays for features, not coverage

Code coverage details what percentage of your codebase is covered by the tests you've written. This can be calculated by the test framework by use of a "code coverage driver" like "pcov".

While you potentially could test 100% of the codebase (those lines of code have to be doing *something*, it might not be feasible or worthwhile to reach 100%. Ultimately, the product has to work and provide a service / tool to the end user so the majority of the development time should be put into achieving that goal.



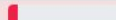





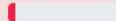





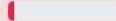
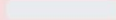
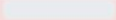
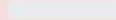




Certainly be mindful of the areas that you need more tests, but I'd be hesitant to make writing tests the priority for all developers!

Code coverage - my first measurement

	Code Coverage								
		Lines		Functions and Methods			Classes and Traits		
Total		70.18%	2182 / 3109		49.88%	212 / 425		3.12%	2 / 64
■ assets		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0
■ controllers		62.79%	432 / 688		56.31%	58 / 103		0.00%	0 / 16
■ helpers		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0
■ models		60.93%	616 / 1011		50.50%	153 / 303		5.26%	2 / 38
■ modules		0.00%	0 / 67		0.00%	0 / 16		0.00%	0 / 8
■ views		84.13%	1097 / 1304		n/a	0 / 0		n/a	0 / 0
■ widgets		94.87%	37 / 39		33.33%	1 / 3		0.00%	0 / 2

70% coverage overall is pretty good, especially given I came to the testing space very late in the development cycle.

Code coverage - my current measurement

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		75.99%	2506 / 3298		54.37%	249 / 458		8.96%	6 / 67
components		44.86%	83 / 185		40.00%	18 / 45		25.00%	2 / 8
controllers		65.20%	459 / 704		59.41%	60 / 101		6.67%	1 / 15
core		n/a	0 / 0		n/a	0 / 0		n/a	0 / 0
helpers		100.00%	6 / 6		100.00%	1 / 1		100.00%	1 / 1
models		74.95%	787 / 1050		55.96%	169 / 302		5.56%	2 / 36
modules		0.00%	0 / 20		0.00%	0 / 6		0.00%	0 / 5
views		87.64%	1134 / 1294		n/a	0 / 0		n/a	0 / 0
widgets		94.87%	37 / 39		33.33%	1 / 3		0.00%	0 / 2

In this second screenshots there's directories being monitored - some new ones have been added and redundant ones have been removed.

The screenshot shows 75.99% coverage overall, which Codeception considers good.

"Modules" clearly needs some work! In this case, I know there are areas in there that should be tested, as a recent change caused a third party integration module to fail - something that went unnoticed by the system owner for months. I should have allowed for that by adding an appropriate test.

Mutation tests

- I've only dabbled with these
- Mutation tool (e.g. *Infection*) adjusts your code to see if your test still passes
 - If the test still passes then your test likely needs adjustment
- Useful for improving your test writing

First the mutation tool checks all your existing tests are passing. Then the mutation tool changes your code, and it's important to note that it's your code that gets changed by the mutation tool, not the test.

If a test still passes with the modified code (for example changing a "greater than 17" to "equals 17") then something is wrong with your test, and you should fix it. This would be called an "escaped mutant".

At the moment I'm only looking at mutation testing from an academic perspective.

The mutation tool reverts any changes it made to your code.

Conclusion

- Using a test framework will save you time
- Gives confidence that your recent changes didn't break something
 - Only if you're testing the right thing...
- Can help assure auditors, investors, and security teams that you aim to develop securely