Code reviews and Quality

MODIO
Secure & efficient energy information

You can fix irreducible complexity by firing everyone who understands it. Then you have a black box.
-- @SwiftOnSecurity

SCM History

Why is history important?

History

```
Issue: #18 …
TestZabbixItem added …
Separate: test_construct_LCDText_with_illegal_name
Test cases: …
Test cases to test so gzip is last
Issue: #61 …
Test-cases for new stricter name validation of check_frontend
Issue: #61 …
Test-cses for new stricter name validation of check_frontend
Merge branch 'isse-61-testcases-rebased' of https://github.com/MyTemp… …
Test-cases for new stricter name validation of check_frontend
c5ccda0
Stricter validation of check_frontend name
Separated validation of frontend
```

(That was a real-world example)

Rules for a good commit

One commit == One change

Commits are Cheap

Good commit messages

http://chris.beams.io/posts/git-commit/

Good commit messages

Commit messages, briefly

- Subject < 50 chars and Capitalized
- No period on the subject
- Imperative subject mood

Commit messages, briefly

- Body wrapped at <72 chars
- Body explains WHAT and WHY
- Code explains HOW

Read the article:

http://chris.beams.io/posts/git-commit/

Who knows SQL?

Hand up!

```
commit 8ddd9f8573c0ffee57d057babe2c74a01205835c
Author: D.S. Ljungmark <ljungmark@modio.se>
Date:   Thu Jul 23 11:56:14 2015 +0200

Get the data

+    def data(cls):
+        query = “SELECT (c).csr_id, (c).not_before, (c).not_after
FROM (SELECT (SELECT c FROM certificate c WHERE c.csr_id=csr.id
ORDER BY c.not_after DESC LIMIT 1) AS c FROM csr offset 0) s;”
+        return Sess.query(query)
```

```
SELECT (c).csr_id, (c).not_before,
(c).not_after FROM
   (SELECT (SELECT c
         FROM certificate c
         WHERE c.csr_id=csr.id
         ORDER BY c.not_after DESC
      LIMIT 1)
         AS c
         FROM csr
         offset 0) s;
```

Was that a good commit?

How should it have been done?

Review happens on a set

0: Review the set

- Does it apply
- Is it coherent
- Is it ready

1: Review the Commit

- Good description?
- Seems complex?
- Diffstat
- Scary code paths?

1: Review the Commit

Does the label on the tin match the content?

Are the comments unchanged? Why?

2: Review hunks

- Coding style
- Names
- Singletons
- Argument order

3: Review functions

- Black box behaviour
- Test cases
- Validation / Contracts
- Find callers

4: Understand why and how

- Does it match the issue?
- Future?
- New interfaces?

5: Is it in the right place

- Right module?
- Right interface?
- Right context?

6: Review comments

“If something except” - Abstraction error

“when” - flow error

Function basics

Extraction

Validation

Processing

Extraction

Parse URL, input, data-stream

Validation

Bounds check

Syntactical / Semantical

MODIO
Secure & efficient energy information

Processing

“Good data” do things with it

Databases are User Input

Previously non-validated data may exist

Assume data _from_ db is toxic

Databases is User Input

MySQL loves the NULL

Code review tools

Find one that works

GitHub, Gerrit, Crucible, Kallithea

MODIO
Secure & efficient energy information

Doing Terrible things to your code

Read:

http://blog.codinghorror.com/doing-terrible-things-to-your-code/

STUPID / SOLID

http://williamdurand.fr/2013/07/30/from-stupid-to-solid-code/

MODIO
Secure & efficient energy information

STUPID

Singletons

add “the” to all singletons

Avoid global state

STUPID

Tight Coupling

Pass instances

Wrap data in data-containers

STUPID

Untestability

Small, testable pieces

Reduced state

STUPID

Premature Optimization

th rt f ll vl

MODIO
Secure & efficient energy information

STUPID

Indescriptive Naming

foo(thingie, things, majing)

STUPID

Duplication

DRY, KISS

SOLID

Single Responsibility Principle

(Avoid God Classes)

SOLID

Open / Closed Principle

Open for Extension

Closed for Modification

Private by default

SOLID

Liskov Substitution Principle

(Subclasses can replace parent, without change in behaviour)

SOLID

Interface Segregation Principle

(Many specific interfaces better than generic)

Coupling, and reducing complexity

MODIO

Secure & efficient energy information

SOLID

Dependency Inversion Principle

Abstractions should not depend upon details

Details should depend upon abstractions.

Look out for one-timers

Single use functions are good for testing

Reduces scope

But hides complexity

As an exercise, inline

Will show which parts of code are scary