Write Beautiful Code

Laura Thomson
laura@laurathomson.com

This presentation is about writing:

beautiful

practical

sleek

code

History / Rationale

I used to give a presentation about bad code.

This is the optimist's version.

It's also at a relatively high level.

<me>

<me>

PHPer for a long, long time

Engineer, consultant, trainer, manager of engineers

Opinionated (Australian)

Work at Mozilla (previously OmniTI, RMIT)

Wrote some books

Principles

Your PHP should

- be simple
- get the job done
- be secure
- be scalable and performant
- be producible and maintainable

Simple

Rule #1

First, do the simplest thing that could possibly work.

Implications

The simplest thing that could possibly work is usually pretty fast to produce.

It may lack robustness, performance, and security.

(Strangely, very simple code is usually simple to maintain.)

Architecture

Many people like big elegant architectures. These are sometimes referred to as “enterprise”.

If the task is simple, build a simple solution.

If the task is large and complex, have an architecture but prototype the hard parts first with small, simple code. This lets you focus on getting it working.

Failure

I see projects fail when people spend all their time on building big and run out of time to make it work.

I see projects fail when inexperienced architects build huge, far sighted architectures and forget to also think small.

Robust

Robust

Robustness means two things:

- getting the job done.
- continuing to get the job done as much as possible even when things go bad.

Testing ?

I don’t just mean testing.

I mean building your code and environment to degrade gracefully in the event of failure, and to resume working later (regrade).

Robust Code

Strategies for robust code include

- Testing common cases and edge cases
- Removing dependencies where possible - especially dependencies on uncontrollable variables (outside APIs et al)
- Decouple code so each piece is itself simple and has an established interface.

Robust Environment

Robust environment?

- Repeatable process
- Lightweight documentation
- Documentation of changes
- Good use of version control

Environment

Minimum: stage/test + production

Preferred: dev + stage/test + production

Need different boxes (virtual++)

Testing

Bare Minimum: use case testing

Good: use case testing + unit tests

Better: human use case testing + full automated (unit, regression, system, load)

Testing = tools + discipline/process

Tools

PHPUnit http://phpunit.de/

SimpleTest http://simpletest.org/

Selenium http://selenium.openqa.org/

Perl

Secure

Most times security is something tacked on at the end of the day

The ideal is a secure design: architecture with a single entry point, et al.

Simple independent code components and a low level of trust is easier to scale

Trust no one

$_GET, $_POST, $_REQUEST, $_COOKIE

Some server variables (e.g. $_SERVER ['SERVER_NAME'])

API data

Files

Girl Scout Cookies

Sleepless nights

XSS (Cross Site Scripting)

SQL/Command/Code Injection

Exposed source code

Session fixation / Session hijacking

CSRF

Cross domain Ajax requests

Zero day

People stay up worrying about the next exploit, when the truth is most websites have plenty of existing problems.

Principles

Peer review new code

Code should not trust other code

Audit existing code

Stay up to date

Don’t depend on external factors for security (where possible)

Make time in your process for security

Have a disaster plan

Scalable
Performant

Scalability vs Performance

Scalable code handles a change in load acceptably

Performant code handles today's load acceptably

Confusion

Confusion arises between these two terms because typically we deal with the issue after the horse has bolted.

Coding for scale

More about what you shouldn't do

Question design decisions:

- What will this mean when there are 100 servers?
- Will this code work if we change the underlying architecture?
- How do I scale-proof my code?

Performant

What makes a good programmer?

“The ability to trust empirical evidence over instinct.”

For performant code we need to measure, tune, and measure again.

Principles

Keep parts of the system as independent as possible

Measure, measure, measure

Focus on the worst thing

Change one thing at a time

Measure again

Independence

If you keep components independent, you can change things out or add more firepower as load increases

- More web servers
- More database slaves
- More caches in more places

Caching

Use a compiler cache (Zend, APC, eAccelerator, etc)

Cache generated content (note: serve static content separately)

- Whole pages or fragments (disk, squid, memcache)
- Database query results (query cache, memcache)

Measurement

You need tools.

- Overall load: httperf, ab, Grinder
- Profilers: xdebug, APD, Zend
- System tools: strace, dtrace, ltrace
- Database tools: (e.g. MySQL) EXPLAIN, slow query log, mysqltop, show processlist

No assumptions

Measure to see where the problem is

Remember diminishing returns

No premature optimization (worst use of developer time ever)

Performance testing

Need to test your architecture? Use production data

New systems with unknown traffic patterns are much harder to test

Caveat: load testing writes is harder

Producible
Maintainable

Producible: able to be created in a reasonable amount of time

Maintainable: able to be updated in a reasonable amount of time

These forces often pull in different directions.

The usual problem.

Don't forget "debuggable"

Using

- output buffering
- framework magic
- black box components

makes debugging much much slower.

Frameworks

(By which people mean MVC)

Suitable for UI apps.

Don’t buy you simplicity or performance.

Some buy you maintainability at the expense of producibility, and some just the opposite.

Do frameworks speed up development?

Write good code

Above all, code needs to be readable and maintainable

Beware clever code:

- cool design patterns
- obscure performance tweaks
- niche language features

Errors and Exceptions

Turn error_reporting up in dev and display_errors off in production

Use set_error_handler() and set_exception_handler() for top level errors

Whacking all your code in a try…catch block is not a panacea.

Coding standards

Have and use a coding standard

Zend Framework, PEAR, homebrew

Legacy code always a problem

Standard Don’ts

Make the rules awkward and difficult to remember

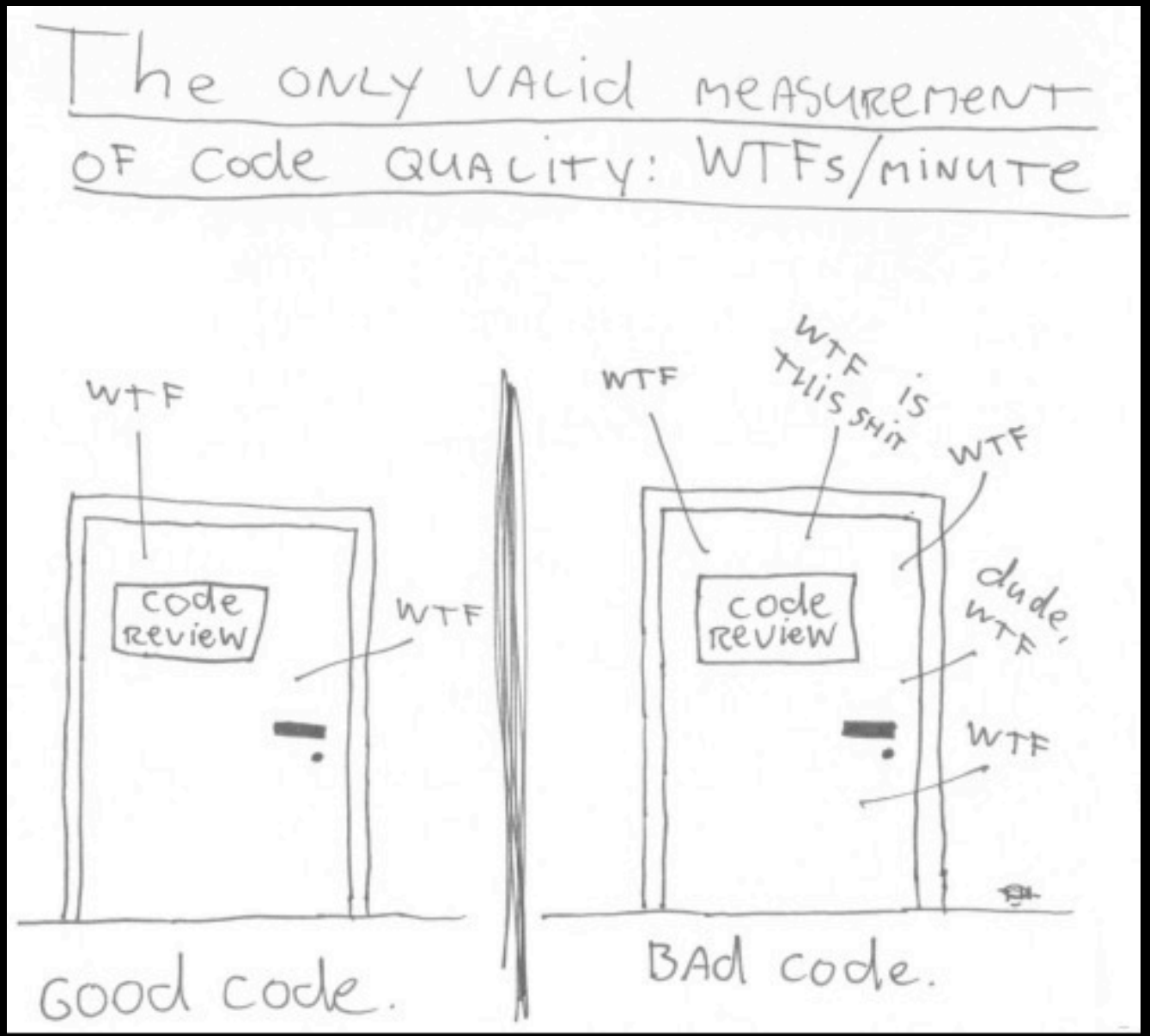
Apps Hungarian

Force millions of include files (performance hit)

Force complete OO (c.f. Simple)

Final words

The goal



(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

Slides

Slides on my blog at www.laurathomson.com

(after the talk)

Questions?