

# How not to release software

Laura Thomson

`laura@{mozilla,laurathomson}.com`

`@lxt`



# O'REILLY OPEN SOURCE CONVENTION.

O'REILLY.COM O'REILLY NETWORK  
CONFERENCES | INTERNATIONAL | SAFARI BOOKS ONLINE

- Home
- Registration
- Speakers
- Keynotes
- Tutorials
- Sessions
- At-a-Glance
- BOFs
- Events
- Exhibitors
- Sponsors
- Hotel/Travel
- Venue Map
- See & Do
- Press
- Mall List

**Kids  
World**  
"The Kid's World  
program and

From the Frontiers of Research to the Heart of the Enterprise  
**O'Reilly Open Source Convention**  
Sheraton San Diego Hotel and Marina  
July 22-26, 2002 -- San Diego, CA

## Tutorial



### PHP: FSP (Frequently Solved Problems)

[Laura Thomson](#), Mozilla Corporation  
[Luke Welling](#), OmniTI

Track: PHP  
Date: Monday, July 22  
Time: 1:45pm - 5:15pm  
Location: Harbor Island III

In running PHP courses and reading the mailing list we find the same problems and questions come up over and over again. These are the sorts of problems you must learn to conquer to earn your stripes as a PHP programmer and graduate from PHP newbie to guru. In this talk we will take a cook's tour of the most common PHP brick walls and how to climb them.



- ▶ Home
- ▶ Registration
- ▶ Speakers
- ▶ Keynotes
- ▶ Tutorials
- ▶ Sessions
- ▶ At-a-Glance
- ▶ BOFs
- ▶ Events
- ▶ Community
- ▶ Software
- ▶ Exhibitors
- ▶ Sponsors
- ▶ Hotel/Travel
- ▶ Venue Map
- ▶ See & Do

## Embracing and Extending Proprietary Software

*Portland Marriott Downtown, Portland, OR • July 7-11, 2003*

### Tutorial

#### **SOLD OUT PHP and Flash**

[Laura Thomson](#), Mozilla Corporation  
[Luke Welling](#), OmniTI

**Track:** PHP

**Date:** Monday, July 07

**Time:** 1:45pm - 5:15pm

**Location:** Salon H





# OPEN SOURCE CONVENTION

Portland Marriott Downtown, Portland, OR • July 26-30, 2004

[Books](#) | [Safari Bookshelf](#) | [Conferences](#) | [O'Reilly Network](#) | [O'Reilly Gear](#)

- ▶ [Home](#)
- ▶ [Registration](#)
- ▶ [Speakers](#)
- ▶ [Keynotes](#)
- ▶ [Tutorials](#)
- ▶ [Sessions](#)
- ▶ [At-a-Glance](#)
- ▶ [Wiki](#)
- ▶ [BOFs](#)
- ▶ [Events](#)
- ▶ [Exhibitors](#)
- ▶ [Sponsors](#)



## Tutorial

### PHP and MySQL Web Development with Bells and Whistles

[Luke Welling](#), Security Analyst, OmniTI  
[Laura Thomson](#), Mozilla Corporation

**Track:** PHP

**Date:** Tuesday, July 27

**Time:** 1:45pm - 5:15pm

**Location:** Salon 1

 [TrackBack](#)

[Home](#)  
[Invitation to Attend](#)  
[Registration](#)  
[Speakers](#)  
[Keynotes](#)  
[Sessions](#)  
[Tutorials](#)  
[Schedule](#)  
[Open Source Business Review](#)  
[Events](#)  
[BOFs](#)  
[Exhibitors](#)  
[Sponsors](#)  
[Hotel/Travel](#)  
[See & Do](#)  
[Convention Coverage](#)  
[Newsletter](#)  
[O'Reilly Conferences](#)

## Tutorial

### PHP/MySQL Best Practices

[Luke Welling](#), Security Analyst, OmnitTI  
[Laura Thomson](#), Mozilla Corporation

**Track:** PHP

**Date:** Tuesday, August 2nd, 2005

**Time:** 1:30pm - 5:00pm

**Location:** Portland 252

Many web programmers use PHP with MySQL. However, lots of hard working programmers work to tight deadlines, are self-taught, and may never have experimented with some of this combination's newer features. In this session, we will look at best practices for using MySQL from PHP, showing how common web development tasks can be made easier, more efficient, or more secure.

Welling and Thomson discuss when and how to use the following features in the web environment:

- Different storage engines
- The MySQL query cache
- Prepared statements for efficiency and protection against SQL injection attacks
- Transactions
- Full text searching
- Subqueries

This tutorial is aimed at programmers who already use PHP and MySQL together, but want to get more out of the pairing.

## Diamond Sponsors



Computer Associates®



invent

spike  
SOURCE



## Platinum Sponsors

Novell.

## Gold Sponsors



Scaling Mozilla's Websites

Laura Thomson  
[laura@mozilla.com](mailto:laura@mozilla.com)



# Writing Maintainable PHP

Laura Thomson  
PHP Quebec Conference  
16th March 2007

# PHP and MySQL Best Practices

Luke Welling  
<http://lukewelling.com>

Laura Thomson  
<http://www.laurathomson.com>

# Write Beautiful Code

Laura Thomson  
[laura@laurathomson.com](mailto:laura@laurathomson.com)

# PHP

# Best Practices

Laura Thomson  
laura@laurathomson.com  
<http://www.laurathomson.com>

# Rewrite or Refactor

When to declare technical bankruptcy

---

*Laura Thomson (laura@mozilla.com)*

*OSCON - July 22, 2010*

Conference speakers and book authors are people too. Despite our smug demeanor, we also fail.

Conference speakers and book authors are people too. Despite our smug demeanor, we also fail.

Frequently.

This talk sparked by the release-that-shall-not-be named.

This talk sparked by the release-that-shall-not-be named.

It went so badly that the version number was retired.

This talk sparked by the release-that-shall-not-be named.

It went so badly that the version number was retired.

After 24 hours, people started bringing in casseroles.

Thus I present this catalogue of failure, a compendium of things not to do when you want to release any kind of software.

# Methodology

# Agile fail #1

Cynical basis of agile: estimation is intractable

# Agile fail #1

Cynical basis of agile: estimation is intractable

Avoid estimation by working in short sprints

- surely we can estimate the next two weeks, right?

# Agile fail #1

Cynical basis of agile: estimation is intractable

Avoid estimation by working in short sprints

- surely we can estimate the next two weeks, right?

Right?

# Sprint fail #1: rollover

We will build these five thingies/bugs/stories  
in the next two week sprint

# Sprint fail #1: rollover

We will build these five thingies/bugs/stories  
in the next two week sprint

Two are more complex than anticipated

# Sprint fail #1: rollover

We will build these five thingies/bugs/stories  
in the next two week sprint

Two are more complex than anticipated

One is blocked

# Sprint fail #1: rollover

We will build these five thingies/bugs/stories  
in the next two week sprint

Two are more complex than anticipated

One is blocked

Ship two, roll the other three over

# Sprint fail #1: rollover

We will build these five thingies/bugs/stories  
in the next two week sprint

Two are more complex than anticipated

One is blocked

Ship two, roll the other three over

Rinse, repeat

# Sprint fail #2: Mara-sprint

We can't get these five things done. Should we ship with only two?

## Sprint fail #2: Mara-sprint

We can't get these five things done. Should we ship with only two?

If we just push the deadline out a week, everything will be fine.

# Sprint fail #2: Mara-sprint

We can't get these five things done. Should we ship with only two?

If we just push the deadline out a week, everything will be fine.

And then another.

# Sprint fail #2: Mara-sprint

We can't get these five things done. Should we ship with only two?

If we just push the deadline out a week, everything will be fine.

And then another.

And then another.

# Sprint fail #3: Death sprints

We HAVE to get these five things done

# Sprint fail #3: Death sprints

We HAVE to get these five things done

The deadline is an immovable object

# Sprint fail #3: Death sprints

We HAVE to get these five things done

The deadline is an immovable object

We'll just work really really late or all night  
every night for the next two weeks

# Sprint fail #3: Death sprints

We HAVE to get these five things done

The deadline is an immovable object

We'll just work really really late or all night  
every night for the next two weeks

And then the same thing again....and  
again...and again

# Things I've learned

Plan the dimension of flex and choose:

Less functionality in the same time

Flexible timelines or slower velocity

Continuous deployment

Train model

Coding

# Coding fail #1: Build a new bikeshed

Sitting down to code when you don't know  
what you're doing

# Coding fail #1: Build a new bikeshed

Sitting down to code when you don't know  
what you're doing

vs

Analysis paralysis/bikeshedding: getting stuck  
talking through a complex problem, over and  
over

# Coding fail #2: One True Ring

There's one right way to solve this problem

Complex

Going to take a long time

Scope keeps expanding

# Coding fail #3: Over-engineering

Closely related to the One True Ring

# Coding fail #3: Over-engineering

Closely related to the One True Ring  
Beautiful class hierarchy and architecture

# Coding fail #3: Over-engineering

Closely related to the One True Ring  
Beautiful class hierarchy and architecture  
Lovely diagrams

# Coding fail #3: Over-engineering

Closely related to the One True Ring

Beautiful class hierarchy and architecture

Lovely diagrams

It doesn't **do** anything

# Things I've learned

Different people work in different ways.  
Some people deal better with uncertainty.  
Those people should build a prototype.

Do The Simplest Thing That Could Possibly  
Work

Be happy with 60-80% solutions. Iterate.  
Even if you're not a startup, Minimum Viable  
Product is valid.

Testing

# Test fail #1: The Null Hypothesis

Have no tests.

# Test fail #2: Epic confidence

Have some tests, but no idea about coverage.

## Test fail #2: Epic confidence

Have some tests, but no idea about coverage.

Ensures you have a great level of over-confidence going into a release.

# Test fail #3: UR DOING IT RONG

Test the wrong things.

# Test fail #3: UR DOING IT RONG

Test the wrong things.

99% unit test coverage, no load tests.

# Test fail #3: UR DOING IT RONG

Test the wrong things.

99% unit test coverage, no load tests.

Push something out and don't realize that it's too slow, causes data or user loss, until too late.

# Things I've learned

Know your coverage and your limitations

Use CI

Use browser testing

Use load and smoke testing

Get QA's opinion on whether you're ready to ship something

**Release and  
deployment**

# Deployment fail #1: Version chaos

Push master/trunk/head.

# Deployment fail #1: Version chaos

Push master/trunk/head.

Have no other branches.

# Deployment fail #1: Version chaos

Push master/trunk/head.

Have no other branches.

Have no tags.

# Deployment fail #1: Version chaos

Push master/trunk/head.

Have no other branches.

Have no tags.

Have things that aren't under version control.

# Deployment fail #2: Fail forward, fail back

Have no rollback plan.

## Deployment fail #2: Fail forward, fail back

Have no rollback plan.

Have no ability to fail forward, either.

## Deployment fail #2: Fail forward, fail back

Have no rollback plan.

Have no ability to fail forward, either.

If failing forward fails, have no backup plan.

# Deployment fail #3: Manual pushes

Release by manually logging into each machine and updating a checkout.

# Deployment fail #3: Manual pushes

Release by manually logging into each machine and updating a checkout.

This is even better under high load.

# Deployment fail #3: Manual pushes

Release by manually logging into each machine and updating a checkout.

This is even better under high load.

Scaling is something best not talked about.

## Deployment fail #4: ALTER TABLE ADD FAIL

Require manual changes to databases. Have no record of the changes in version control.

## Deployment fail #4: ALTER TABLE ADD FAIL

Require manual changes to databases. Have no record of the changes in version control.

Don't estimate how long these changes will take, how much they will degrade performance, or how much downtime will be required before kicking them off.

# Deployment fail #5: Config of Doom

Require manual changes to configuration, especially if there are more than two machines involved - 30 or more is especially good.

# Deployment fail #6: Single Person Of Fail

Have only one person that knows how to  
deploy your stuff.

# Deployment fail #6: Single Person Of Fail

Have only one person that knows how to  
deploy your stuff.

(Don't document any of it.)

# Deployment fail #7: It Worked In Staging

Staging not the same as prod:

# Deployment fail #7: It Worked In Staging

Staging not the same as prod:

Different OS, packages, hardware

# Deployment fail #7: It Worked In Staging

Staging not the same as prod:

Different OS, packages, hardware

Only one of something where there are >1  
in prod (classic examples: db replication,  
memcache)

# Things I've learned

Have a release management system

Build the capability to deploy continuously

Automate everything: build, test, deploy

Use configuration management

Document everything

Load test when needed

Eliminate SPOFs, including people

Staging should reflect production

Operations

# Ops fail #1: Devops Fail

Ops and Dev don't speak to, work with, or trust each other

# Ops fail #1: Devops Fail

Ops and Dev don't speak to, work with, or trust each other

Most communication takes place through aggressive ticket filing and email

# Ops fail #2: Dilettante devs

Stuff breaks at 2am

# Ops fail #2: Dilettante devs

Stuff breaks at 2am

Devs have not documented anything and  
aren't available to fix it

# Ops fail #2: Dilettante devs

Stuff breaks at 2am

Devs have not documented anything and  
aren't available to fix it

Ops gets the blame for extended downtime

# Ops fail #3: Telepathic troubleshooting

Stuff breaks

## Ops fail #3: Telepathic troubleshooting

Stuff breaks

Devs have to troubleshoot via remote:

Did you look at this log, what did it say?

What is this config value set to?

# Things I've learned

Everybody who works on a system is responsible for it

Cross train your teams

Open up systems to devs for read access to config + logging

Troubleshoot together

Questions?  
[laura@mozilla.com](mailto:laura@mozilla.com)  
PS We're hiring