



# Large Scale PHP: Devteam Infrastructure

Jonathan Oxe  
November 4th, 2004: phpMelb

# Large Scale Project Infrastructure

#1: Source Code Management Systems

#2: Bug Tracking Systems

#3: Internal Documentation

Tie it all together to create a semi-automated development infrastructure to save your developers from boring and repetitive tasks.

# #1: Use Source Code Management

Source Code Management systems (SCMs) rock. Definitely the single most useful tool for a development team, ranking second only to a good text editor (just!).

SCMs allow multiple developers to collaboratively work on the same codebase and tracks which changes have been made by which developers. They also attempt to automatically merge changes together when possible, and provide a mechanism to assist with manual merging when necessary.

Well known SCMs include CVS, Subversion, ARCH, BitKeeper and Visual SourceSafe.

# SCMs: Centralised vs Decentralised

SCMs generally fall into one of two categories: centralised and decentralised.

Centralised systems use a master “repository” with all developers working on a checked-out “working copy”. All communications is between the repository and the working copies.

Decentralised systems have no one master repository, being structured more like a peering network of working copies.

Newer SCMs are tending toward being decentralised.

# SCMs: Changeset vs Patch Oriented

SCMs also generally fall into one of another two categories: changeset or patch oriented.

Changeset oriented SCMs work in terms of “revisions” or “versions” of the source tree or files, and changes are sequential.

Patch oriented SCMs work in terms of discrete “patches” which are applied to the codebase. Developers can choose to apply certain patches and discard others.

Patch-oriented SCMs are a relatively recent development and less developers are familiar with them than with changeset SCMs.

# Which SCM To Choose?

Not quite a religious war, but developers often have strong preferences. CVS is very widely used and many developers are familiar with it but it's quite dated.

Subversion is rapidly taking over as the most widely used SCM in the F/OSS world: it's essentially a “work-alike” re-write of CVS, so many developers already have the workflow wired into their hind-brain.

ARCH-based systems are very interesting if you're willing to go to just a tiny bit more effort to get your brain around it.

For an overview of SCMs see [better-scm.berlios.de](http://better-scm.berlios.de)

# #2: Bug Tracking Systems Rock

A Bug Tracking System (BTS) can help you assign, prioritise and report on tasks and bugs.

A BTS is really just a pumped-up To Do list but it can dramatically simplify the process of managing your projects, and also provide an invaluable central location for visible collaboration.

# Bug Tracking Systems Suck

A BTS is just a funky To Do list, right? So they're simple, right? And of course that means they're easy to install?

Think again.

You don't truly appreciate the word “frustration” until you've tried to deploy Debbugs or Bugzilla.

# Features To Look For

Different people and groups have different requirements, so consider how you prefer to work.

- Web interface
- Email interface
- External hooks
- Access control: users and groups
- Categories and item types
- Reports
- Comments
- Bug dependencies
- File attachments

# Shameless Plug: Flyspray

Flyspray is a BTS written in PHP and using either MySQL or Postgresql as a back-end. Quite apart from its functionality as a BTS it has some interesting internals, such as use of ADOdb for database abstraction and multilingual support using language packs.

It's designed for ease of installation and simplicity for first-time users.

<http://flyspray.rocks.cc>

For more information see *Killing Bugs With Flyspray*:

<http://jon.oxer.com.au/flyspray>

# #3: Internal Documentation

“Documentation” is a four letter word, at least the way developers think about it and spell it: docs. But on a medium to large project they can make your life so much easier for a couple of reasons.

Firstly as a memory refresher for yourself and your teammates.

Secondly when bringing new developers onto a team.

# Auto-generated Docs

Creating internal documentation doesn't have to be a great effort or require you to assign someone to write it.

Be lazy like a fox: set up PHPDoc and let your system build the docs for you. Check out

- PHPDoctor ([phpdoctor.sourceforge.net](http://phpdoctor.sourceforge.net))
- phpDocumentor ([www.phpdoc.org](http://www.phpdoc.org))
- phpdocgen ([www.arakhne.org/phpdocgen/](http://www.arakhne.org/phpdocgen/))

Then just make sure everyone puts properly-formatted comments in their code. And we all do that already of course! ;-)

# PHPDoc Comment Formats

The various PHPDoc implementations all parse the comments in your code to create their docs. They use a “DocBlock” as their standard unit, and they look a little something like this:

```
/**
 * This is my groovy function to multiply two numbers
 * @param integer $width The width of the wall
 * @param integer $height The height of the wall
 * @return integer $area The total area of the wall
 */
function find_area($width, $height)
{
    $area = $width * $height;
    return $area;
}
```

Not too hard is it?

# Tying Systems Together

The tools I've been talking about are fun on their own, but things get really interesting when you tie them all together into an integrated workflow management environment.

Start by looking at the “hooks” provided by your SCM, since they will be the glue that binds your environment together. For example, Subversion provides “pre-commit” and “post-commit” hooks.

Then look at the hooks provided by your BTS.

Then think about how you can link everything together with scripts and helper programs and bits of fencing wire.

# Automation 1: Changelog Emails

Use your SCMs “post-commit” hook to send an email to your devteam mailing list whenever a commit is made.

The email can contain the name of the developer who performed the commit, the files that were altered, and the text of the changelog entry.

# Automation 2: Bug Closures On Commit

When developers commit new code to the repo it often fixes a bug, so then the bug itself has to be closed or tagged in the BTS.

Why make developers do extra work? Close it automatically!

Use a post-commit hook that scans the changelog for entries like:

Closes: #123

then connects to your BTS and marks that bug as closed and appends the changelog entry as a comment.

# Automation 3: Rebuilding Docs

Put your automatically generated internal documentation on a web server somewhere that can be accessed by your dev team.

Then create a post-commit hook that updates a local copy of the source tree then re-runs PHPDoc to update your internal documentation.

Look ma, no hands!

# Automation 4: Automatic Unit Testing

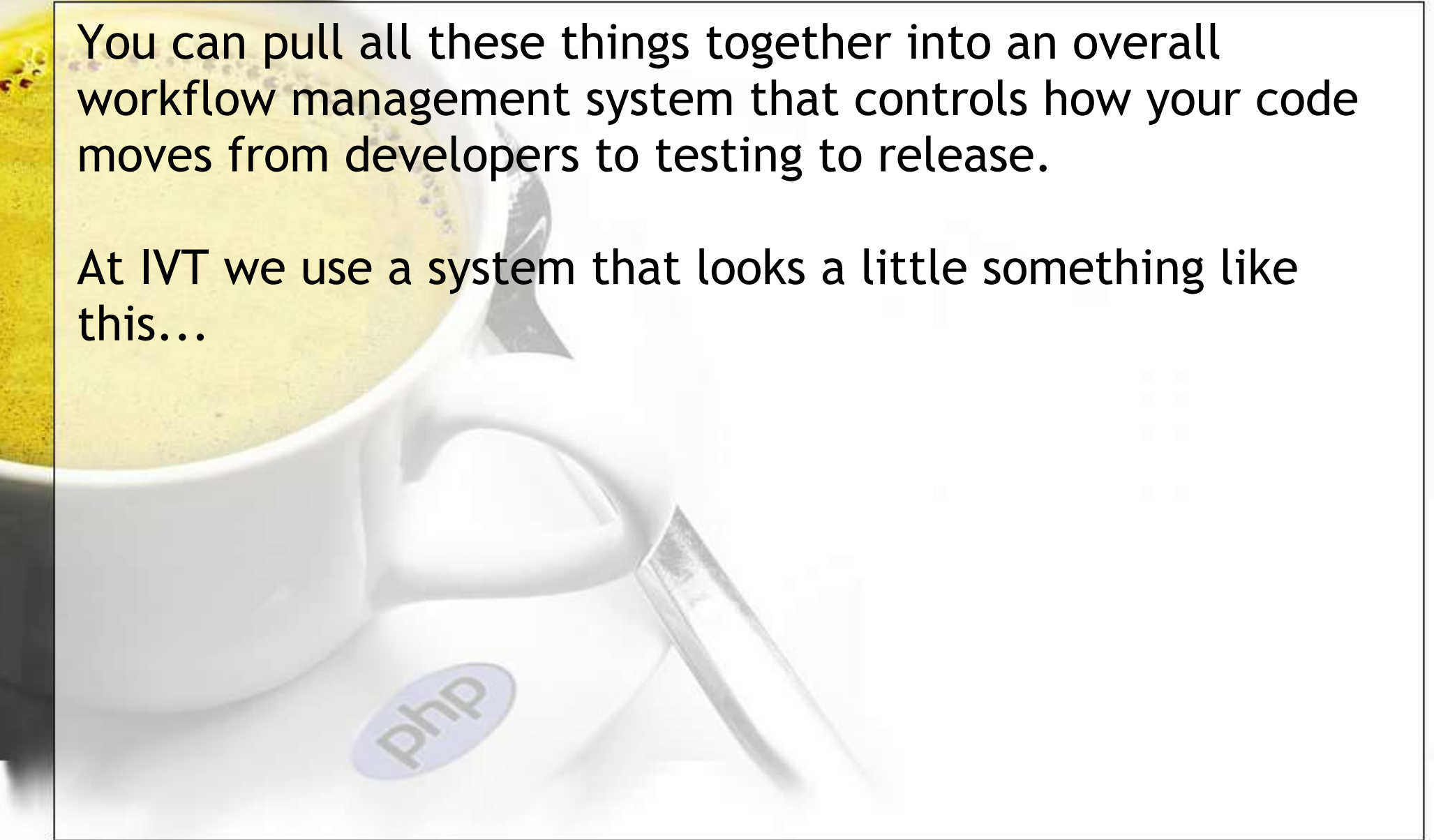
If you have a test mechanism that can check specific functionality of your code, set it up on a test server somewhere then have it triggered on - you guessed it - a post-commit hook that updates a local working copy before running the tests.

Any test failures can then be emailed to your devteam or the developer who committed the change that caused the breakage.

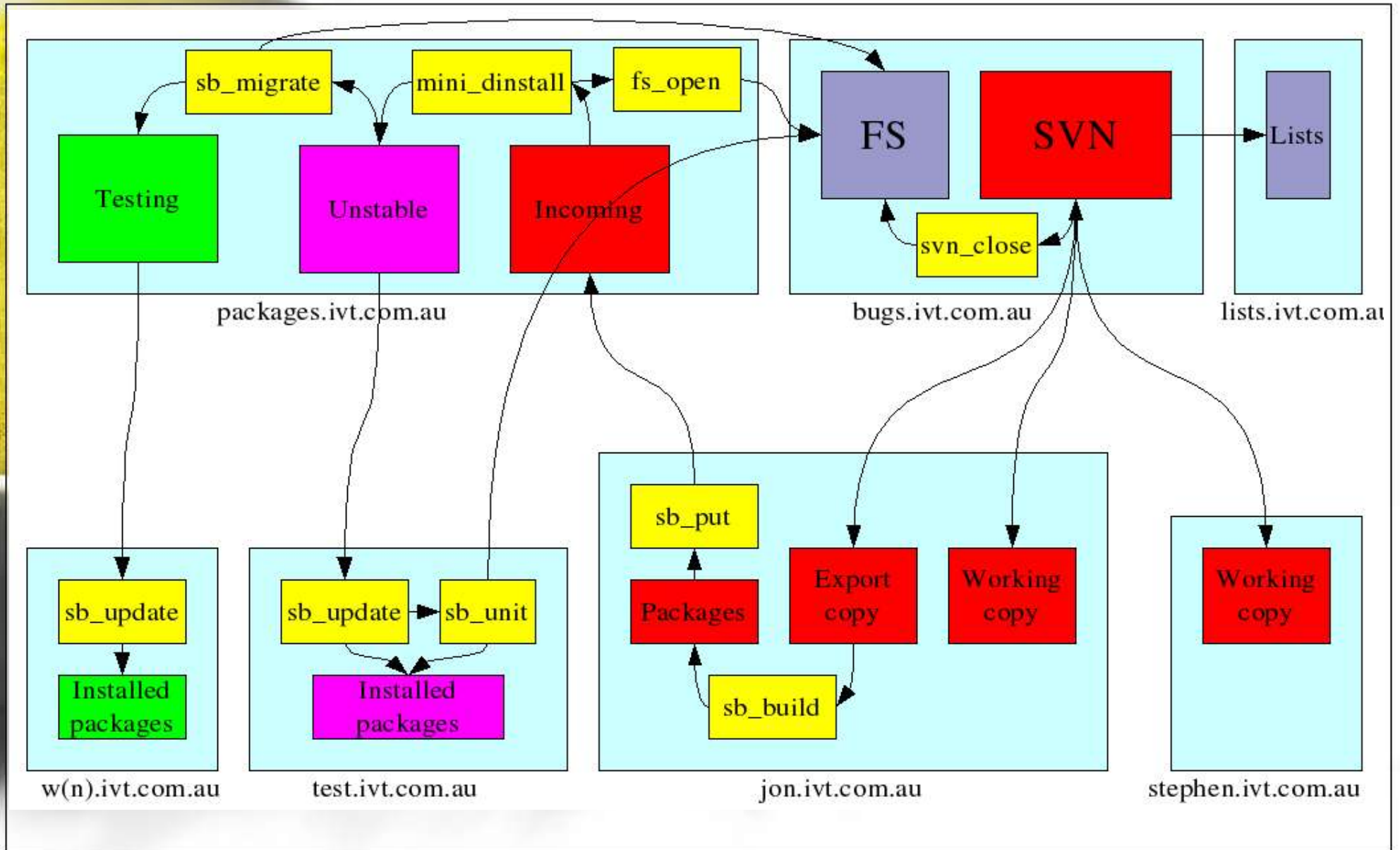
# Automation 5: Release Management

You can pull all these things together into an overall workflow management system that controls how your code moves from developers to testing to release.

At IVT we use a system that looks a little something like this...



# Automation 5: Release Management



# Things I Forgot

Of course working as part of a productive team has many more elements to it than I have mentioned here:

- Coding standards
- A development methodology
- Good project specs
- A good team leader to take client heat for you
- Table tennis at lunch time

# More Information



These slides are at [jon.oxer.com.au/talks](http://jon.oxer.com.au/talks)

If you come to OSDC you'll also get a printed conference paper for this session, so sign up now at [www.osdc.com.au](http://www.osdc.com.au)