Development Process: A Checklist

Erik Hollensbe

Copyright © 2008 OpenSourcery, LLC

Table of Contents

1 Introduction

This is a detailed list of procudures when handling a certain set of situation that are crucial to the forward development of the eleMentalClinic application and its framework.

The expectation is that deviating from these processes is the **exception**, not the **rule**. Therefore, it is a contract between the maintainer(s) of this document and the developers to follow this documentation and ensure its accuracy to current best practices.

2 All Development

These practices apply to all aspects of development.

2.1 Testing

Making any modifications to the application follows this workflow:

1. Test the case
2. Watch the test fail
3. Fix the code until the tests pass

The tests should adequately test problems with the application and expose any bugs that may be relevant to this cycle of development. That is, fixing the code that causes the tests to pass should also fix the bug by side effect. If this is not the case, repeat steps 1-3 until it does.

Note that views are currently hard to test accurately in our environment. Also, legacy controllers cannot be tested. However, both new controllers and all models can be tested.

2.2 Method Definitons

All methods should have a leading and trailing line marker. It looks something like this:

```
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Or a deviation thereof.

All methods should have a snippet of POD (**not comments**) describing, at minimum its input and output. If the method is cryptically named or complex in logic, a description of what the method does should be in order. Whether that method should be renamed or its logic should be refactored is an exercise left to the situation, and mostly, to the reader.

2.3 File Headers and Footers

File headers should have a description (in POD) of the class/package implemented and the purpose of it.

The module return value for the purposes of import() (the literal 1 at the end of most perl packages) is `'eleMental'`. Don't gripe how this is deprecated in 5.8. This is historic and flavorful. It goes good with steak.

Copyright and Author information goes at the bottom. Please use the existing template found in many other files. If you made edits, please add your name and OpenSourcery email to the list.

3 Themes

These processes apply to development of Controller/View themes, such as Default, Mercury, Venus, and Earth.

3.1 Guidelines for writing new views

Randall, can you fill this in?

3.2 Guidelines for writing new controllers

First, let's see if you need to write a new controller. Follow the flow here, and if you answer *No* to any of the **(stop)** segments, you probably shouldn't be adding a new controller.

1. Does it create new functionality not covered in Default or the current theme? **(stop)**
2. Is this theme Default?
   - If so, are we intending to add this to all themes? **(stop)**
   - If not, are we doing specific functionality for a theme? **(stop)**
3. Can we add this functionality elsewhere? **(stop)**

If you do need to add a new controller, here are some things that are **required** for new controllers:

- It must be testable (use the return syntax instead of doing the template processing)
- It must have tests before method bodies exist
- It must leverage ValidOp constraints.

Try to clone and modify an existing controller if you're just bringing slightly new functionality to a theme from Default.

3.3 Porting and Change Propogation for Models and Migrations

If you make changes to a model or migration, these things should be handled:

1. Run the test suite
2. Write controller tests for items that may be affected by the change
3. Make any changes to theme controllers
4. Ensure that those changes make it into Default.

It's **extremely** important that Default is stable – it's the basis for all other themes.

4 Framework

These processes apply to `::Base`, `::CGI`, `::DB`, the scripts in `bin/`, and anything with the word `Dispatch` in it.

4.1 Achtung! Stop!

Be very aware that you are making a global change to the application. The framework is a very sensitive part of the code and your changes will affect every iota of code that powers the application, no matter what part of the framework you touch.

That said, use your best judgment when making changes and really examine whether or not the change you're making belongs in the framework, or MVC code.

4.2 Scripts

These predominantly live in `bin/` and have utility to the code, but are not a part of the code themselves.

Scripts should share in the UNIX tradition and *do one thing and do it well.* They should also have these components:

1. A help output accessible via `-h` and `--help`
2. Accurate Help
3. Proper use of STDOUT and STDERR (for inclusion in Makefiles)
4. Proper use of the environment (e.g., should have a command line switch to override environment and should not need to be changed frequently)

4.3 Dispatcher

The dispatcher is special because it gets at the core of the application and must be developed in pure `mod_perl`. The dispatcher cannot use any of the eleMentalClinic framework or application logic.

That said, the dispatcher must:

1. Target our current Apache requirement (as of this writing, that's 2.2.x)
2. Target our current recommended configuration.
3. Target our current mod_perl and Perl environment
4. Not break, under any circumstances.

Items 1-3 are covered in the versioned `INSTALL` document.

Regarding #3, the dispatcher should be able to handle several hours of fuzz testing without breaking anything in the Dispatcher itself.

4.4 Framework libraries

`::CGI`, `::Base`, `::DB` are libraries that control the Framework.

Framework modules should have:

- Code that drives the rest of the application in a application-agnostic fashion.
- Nothing else

That is, `::Base` should not have application logic in it, and `::DB` should not expect any information about a schema to exist. Likewise, `::CGI` should not know about a specific controller's parameters.

5 Models

These processes apply to development of damn near everything in the lib/ directory that isn't covered by "Framework" above.

6 Migrations

These processes apply to everything in the `database/schema` directory.

6.1 Achtung! Stop!

Modifying existing migrations can compromise the upgrade path for an application. Don't do it, ideally ever.

Also, take great care in creating migrations that run before a pre-existing one. This is preferable to modifying an existing one, but keep in mind that you are doing the same level of damage to the upgrade path. This should never be done in stable/frozen trees.

6.2 Creating a new migration

Creating a new migration works like so:

1. Create a new, numbered migration that follows the others
2. Fill it with SQL

It's is **OK** and in fact **preferable** that you use a migration to undo an old migration instead of editing the old migrations. In otherwords, when faced with a situation where a migration will change something that a previous migration changed, **do not** change the old migration. Create a new one that migrates the already-migrated information.

6.3 Testing migrations

Migrations should be tested against a clone of a production database before any kind of deployment happens. Coordinate with a client or a lead developer to ensure this happens.

7 Old Versions

These processes apply to development on previous versions of eleMentalClinic.

7.1 Fixing a bug in two versions

When we have to backport a fix, follow these steps:

1. Follow the general development logic in the earliest version first
2. Follow the "Porting Changes" logic below

It's important that the fix gets applied in the earliest version first because the logic will progress as the versions change, and modifications will cascade but may create more regressions as we progress up the version list.

7.2 Porting changes

This is the process for porting changes:

1. Port your tests
2. Run the tests suite and ensure it fails
3. Port your changes
4. Run the test suite and ensure it passes
5. If you are porting a controller which has tests in a new version, write controller tests

These steps ensure that the changes get equally applied to all versions.

7.3 Identifying Ports

In Trac, please ensure that all revisions in a port go in the log so that the ticket owner and the administrators are aware that multiple commits were applied to differing versions of the application.