Toolkit implementation: Technical options report

Martin Lucas-Smith, Version 1, 18th August 2011

This report outlines the various options for implementing the toolkit. Comments are very welcome.

Issues

Issues which govern the choice of technology are:

- **Existing code**: CycleStreets has a body of existing code which could be used for some of the functionality; for instance, the instance domain switching system (written as a PHP class) is well-developed and would need very little change. That said, however, existing code which has been subject to extensive use over the years could be translated to another language or framework with little difficulty, since the core logic embodied within the code is what is important to retain. Some of the code, such as the Photomap code, is in need of substantial refactoring.
- **Cost**: A total budget of £27,000 is available. This has to cover the main development work (server-side and client-side), design, information architecture, and a small amount for hosting. This figure is very tight (it is below the budgeted amount) and so additional funds are being sought from another grant source to increase flexibility.
- **Ongoing maintenance**: Obscure forms of technology present a barrier for continued long-term maintenance. The project is intended to be created by an intensive 3-month period of development, after which it is proposed that the source will be maintained on a long-term basis by interested people within the cycling campaign community who have the necessary web development skills.
- **Knowledge of existing CycleStreets personnel**: The current CycleStreets site is built in PHP using 150 or so discrete classes. A PHP-based framework therefore has the advantage of making the project easier to manage, as the benefits and limitations of the language are better-known to those managing the project.
- **Code quality**: Some development languages/frameworks lend themselves to higher-quality code and require people with a more advanced level of experience and understanding.
- **Quick prototyping**: It will help reduce risks in development if functional but unpolished interfaces can be developed quickly up-front. Polish and additional functionality can then be built around a basic proof-of-concept prototype that appears to work well.

Server-side code: choice of technologies

There are a choice of technologies, outlined below.

Key issues for a particular technology are marked with *

(i) Discrete PHP library -based development

This is basically the same model as the current CycleStreets site, namely the development of

custom classes written as PHP.

Advantages:

+ Easy to embed in other contexts – no dependencies whatsoever
+ Reuse of CycleStreets code (Login, Photomap, Journey planner, form library, domain switching) though some of these need rewriting anyway, and some could be translated without too much difficulty
+ Easy to find developers
+ Existing CycleStreets developers know this very well
+ Very heavily used internationally: excellent support
+ Excellent documentation on PHP specifically
+ Fast as no framework overhead
+ Clearer view of what the system actually does – no abstraction
- Requires work to get basics in place (though most work is non-generic business logic)
- Developers mixed: not always best quality
- No standardised coding style
- No automated test-driven development (which is a problem on the CycleStreets site itself)
- Libraries are often of poor quality and hard to find, though things like Zend Framework have libraries that can be used reasonably standalone. Libraries often poorly documented and/or need lots of patching.

(ii) Zend Framework or a similar PHP framework

Zend Framework is a PHP framework intended to improve the development of web applications. It arguably takes a very Rails-like approach to development.

+ Good quality, Rails-like approach, with benefits of code structure, good libraries, etc.
+ Implies better-quality coders
+ MVC approach
+ Enforced naming conventions
+ Very clear documentation
+ Large community
- Existing CycleStreets have no experience in this
- Some criticism of speed due to many queries being run
- Like any framework, has a learning overhead and abstraction
- Developers more scarce as this is more specialised than generic PHP development

(iii) Drupal (PHP)

Drupal is a more traditional CMS-like system, written in PHP, but is more framework-orientated

compared to other CMS systems. It has a large number of pre-available modules.

+ Modular architecture is a good design for reusing shared code from external sources
+ Powerful user/role/permission system
+ Powerful Views and Rules modules enables quick prototyping
+ Well-tested open source code available covering Polls, User registration systems, Content rating, etc.
+ Good security reputation, except for Themes
+ Installation on third-party systems (e.g. directly on campaign sites) easy via modules, though this loses the national sharing context intended for the project
+ Flexible system for allocating different styles to different domains
- Existing CycleStreets have no experience in this
- Some criticism of architectural model from the Rails/MVC camp
- Common view in blog discussions seems to be “Drupal is a ultimately a CMS, Rails is a framework”.
- Developers more scarce as this is more specialised than generic PHP development

(iv) Ruby on Rails

+ MVC approach enforces good quality code structure. Separates out the development roles within a team well.
+ Impressively clean architecture.
+ Quick prototyping. Also attempts to automate the drudgery of web development.
+ Good libraries, e.g. IMAP processing
+ Coders likely to be good, as Rails is less suited to lower-quality developers
+ Same as used by MySociety for similar projects: could be code re-use opportunities there
+ Heavily used within the OSM community
+ Rails seen as a very professional system for good code
+ Test-driven development will assist quality
+ Deployment system safe and well-structured
+ Enforced naming conventions
+ Excellent documentation
+ API-driven approach (API is almost automatically achieved)
- Inability to reuse/share any existing CycleStreets code (Login, Photomap, Journey planner, domain switching) though some of these need rewriting anyway
- Rewriting components such as Journey planner interface feels like a waste of resource, but API is clean so may not take too much time, and lessons learned from CycleStreets can be applied directly without experimentation

- Inability to backport any reusable changes into the CycleStreets code (which would benefit areas like the Photomap which need refactoring or new code)
- Existing CycleStreets have no experience in this other than a day's introductory training
- Cannot be embedded within other systems: is all-or-nothing approach (though the project is a fully-fledged, integrated system, so this does not necessarily apply)
- Bad reputation for scalability but arguably now out-of-date and not an issue for us
- Hasn't taken off in mass popularity like PHP has

Database system

(i) MySQL

+ Currently used by CycleStreets, so opportunities for direct data sharing (though an API approach is probably better structurally)
+ Heavily-used internationally for web applications (but not GIS applications)
- Relatively poor data integrity model
- GIS support relatively poor
- Becoming more difficult to support within CycleStreets (long-term objective to move to PostgreSQL for CycleStreets)

(ii) PostgreSQL

+ GIS support excellent
+ Solid data integrity model
+ Heavily used within OSM community
+ Good command-line support which avoids the need for workaround GUIs like PhpMyAdmin
- Not currently used by CycleStreets

Client-side coding

Only jQuery has been considered. This is the leading Javascript framework.

jQuery

+ Used already on the CycleStreets website
+ Popular (used by over 46% of the 10,000 most visited websites, according to Wikipedia)
+ Well-supported
+ Lots of libraries available
+ jQuery UI (related project) has lots of useful stuff

+ Most browsers will have it cached anyway from various national sites using centralised hosting
+ Good documentation
+ Good browser support and well-tested

General development methodology issues

- Should the system assume that Javascript is available? Some consider this now a core web technology. Alternatively, should we use progressive enhancement techniques, which jQuery helps with?