

Drupal as a Community-Based Software Development Platform

Derek Wright <drupal@dwwright.net> <http://drupal.org/user/46549>

Current status and capabilities

The Drupal development community uses Drupal to coordinate and write nearly all of the code and documentation for Drupal. This “eating our own dog food” is an important aspect of having a real piece of software that solves actual problems. The fact that drupal.org is such a large site helps drive the development of Drupal itself.

In particular, the needs of the Drupal development community in terms of issue tracking, our version control system, release management, access control, and documentation have driven a number of improvements in the Drupal modules related to software development.

Here’s a brief summary of the current status of some of the Drupal modules in this area:

Project – Every module, translation, or theme on drupal.org is represented by a project node. This provides release management, simple project-specific documentation, various ways to browse projects, and as the central entity that most of the other modules work with. Examples: <http://drupal.org/project>
<http://drupal.org/project/drupal>

Project issue tracking – Provides the issue tracking (bugs, feature request, tasks, etc). Numerous ways to sort and display issues on the web, plus optional email notifications of changes to issues,
Example: <http://drupal.org/project/issue>

CVS management/tracker – Integration with the CVS version control system. Imports all CVS commit activity and provides per-user or per-project views of all changes to the source code. Provides CVS account management and per-project access control.
Examples: <http://drupal.org/cvs>
<http://drupal.org/cvs-account>

Book – Core module that drives the Drupal handbooks for online documentation.
Example: <http://drupal.org/handbook>

Limitations

However, even with all of this functionality, Drupal still has a number of limitations that prevent it from wide-scale adoption as a platform to coordinate software development. Many software projects continue to use turn-key solutions such as Trac, which are powerful in some respects, but very limited in others. Here are some of my ideas about Drupal’s short-comings in this area:

Issue tracking

Currently, issue tracking is very rigid. The fields in an issue are fixed, they make a lot of assumptions about exactly what you need, and are usually difficult to customize. Filtering and displaying issues is also inflexible, with a set of hard-coded ways to see the issues. As with most aspects of Drupal, eventually *views* and *dynamic content types* (what used to be called “*CCK*” will be likely solutions to these problems. The ability to dynamically extend an existing content type (issues) to add new fields would be tremendously helpful.

Issues also lack any mechanism for specifying dependencies and relationships. There’s a `duplicate` state that issues can be set to, but there is nothing that automatically records what issue is duplicate with what. Other kinds of relationships are not supported, such as an issue depending on another issue (or set of issues), or a large issue that is split up into individual sub-issues.

The integration with CVS only works in one direction. Developers can refer to the issue number with ‘`#12345`’ in a CVS commit message, and when viewing the commits, this becomes a link to the issue. However, if you view an issue, there’s no way to see all CVS commits that refer to it.

There’s no global view of all activity on a project. You can either view all source code changes, or you can view all issues, or you can view all documentation, but there’s no unified, chronological timeline to see what happened. It would be great to be able to sequentially view commits, new issues or replies, documentation updates, and be able to filter this based on source code branch or user.

Issue tracking (continued)

It is currently impossible to assign an issue to another user. You can only assign it to yourself, unassign yourself, or leave it assigned to someone else.

Email integration with issues is fairly primitive:

- * You can't subscribe yourself to an individual issue such that you start getting email notifications without actually posting a reply to that issue (which then notifies everyone already subscribed, but you haven't actually updated anything or added anything useful).

- * You can't create new issues via email. This might be less of a concern for drupal.org itself, but this is a major requirement for other sites.

There is no way to keep track of the time anyone spends on a given issue. Again, drupal.org itself might not care about this, but to really be adopted by other projects, this will be an important feature to provide.

Releases

Currently, the way the *project* module handles releases is completely dysfunctional. Many sites simply give up trying to use this module since the process for adding new releases to a project is full of bugs and broken assumptions. Even with the forth-coming new release system, there will still be some limitations.

For example, each Drupal release (both now and under the initial versions of the new system) assumes that there's only a single file that comprises the release itself. If a release could include multiple files, we could use this functionality on drupal.org to provide both `.tar.gz` and `.zip` versions of modules. Sites that release binary versions of their software would need to be able to provide separate packages for different platforms.

Tests

None of these modules integrate directly with testing of any kind. We should explore ways to view projects that have tests and their results.

Version Control Systems

All of these tools are tied exclusively to CVS. Some day, Drupal will want to be more flexible and support other systems. Much of the code now in the `cv` module should be abstracted.

Possibilities

Aside from the fact that all of these limitations can be removed (given enough help or sponsorship to generate the code) Drupal provides some unique possibilities for software development communities.

Organic Groups (OG)

OG provides some fascinating possibilities regarding software development projects. At the university-related sites I'm building for my day job, we're planning to use OG to represent groups of developers or users who need to track issues. Each issue could be assigned to one or more groups to represent who cares about that issue. Projects themselves would be global, not necessarily associated with any particular group. The group home page would therefore list all the issues, across all projects, that any given group of people cared about, along with documentation written specifically for that group of users, forums, etc.

Another model would be group-specific projects. We could either attach an issue tracking queue to a group node, or continue to use projects in whole, but assigned to a particular group. In this case, group membership could be used to limit source code access control, provide a pool of users who could assign issues to each other, and so on.

Taxonomies

Drupal's taxonomy system could be put to even better use for certain software development functionality:

- * Every project should automatically create a term in a vocabulary so that any content on the site (especially forums and on-line docs) could be tagged with a specific project

- * Same thing for versions or branches

What are your ideas?

Let's brainstorm now.

Feel free to submit issues against the project module or `project_issue` module.

Please help! ;)

Removing these limitations or adding some of these ideas would be a big benefit for Drupal. But, there's only one instance of dww...