UiB DRUPAL

Report 1

December 2012, by frontkom

UNIVERSITETET I BERGEN

Index

Oppsummering på norsk

Når Universitetet i Bergen skal over på Drupal, er det i høyeste interesse for Norges Drupal-milijø at dette blir en suksesshistorie. Frontkom har fått den ære å inspisere utviklingen som er gjort fram mot pilotfasen, og kan allerede nå røpe at dette lover godt.

Dette er første rapport av tre, men også trolig den viktigste. Under følger en kort og forenklet oppsummering av våre funn, etterfulgt av en mer detaljert og systematisk gjennomgang på engelsk.

Sitebuilding og generell konfigurasjon av Drupal. Utbyggingen av nye UiB.no er utført med dyktighet og nær kjennskap til Drupals "best practices". Valg av moduler og metoder som er brukt både for frontend og backend er bærekraftige og velprøvde. Vi har kun noen få bemerkninger, mest med tanke på forenklet vedlikehold.

Bruk av tredjeparts moduler. Installasjonen har etter vår oppfatning et fornuftig antall utvidelser (ca. 60), og de er gode valg for å løse oppgavene de er der for. Blant disse modulene er nesten alle velprøvde og populære løsninger. For de få som er i betaversjon, oppfordrer vi teamet til å bidra til at disse kan få en stabil versjon på drupal.org.

Kode skrevet av UiB for backend (PHP). Custom kode er velskrevet og ryddig. Det er tydelig at det er et kyndig team som jobber med prosjektet. I noen tilfeller finner vi imidlertid mangler knyttet til Drupal Coding Standards og strukturell separasjon av logikk.

Kode skrevet av UiB for frontend. UiB har valgt et moderne og solid base theme som grunnlag for kodingen av frontend. Vi liker spesielt at de støtter moderne verktøy som SASS og Compass. Frontendkoden som er skrevet av teamet har vi enkelte strukturelle bemerkninger til, men holder ellers god kvalitet.

Ytelse: backendkode. Vi har ikke oppdaget noen flaskehalser i koden som skulle true med å skape vanskeligheter. Et par tips om moduler som forbedrer ytelse foreligger.

Ytelse: frontendkode. Teamet har gjort en god jobb med slank og fin kode, og vi har kun få bemerkninger. I tillegg vil vi også tipse om et par moduler og løsninger som kan bidra til ytterligere bedre ytelse.

Bruk av versjonskontroll. Prosjektet viser en god og profesjonell bruk av Git versjonskontroll. Det er markert tydelig hva hver commit i historikken har utført, og det er tydelig at teamet har gjort et godt arbeid med å holde repoet ryddig. Vi tipser om at nærmere lansering kan Git flow være et nyttig verktøy for videre arbeid.

Deploysystem. Vi savner noe dokumentasjon rundt driftsmiljø og deploysystem, men vil holde fram at infrastrukturen ser fornuftig ut, og at klok bruk av versjonskontroll er på plass som bærende element her.

Realisme i prosjektplan. Teamet har gjort et godt planleggingsarbeid, og vi vurderer forholdet mellom kapasitet, utviklingstid og utviklingsoppgaver som balansert. Ut fra den informasjonen vi har tilgang til, tror vi endelig lansering vil gå som planlagt.

Prosjektverktøy. Utviklerteamet benytter Redmine (åpen kildekode programvare som også er i bruk hos andre enheter i UiB) for å knytte utvikleroppgaver til person og versjon, med tidsestimater og tydelige frister. De gjør dette på en ryddig måte. Enkelte saker kunne imidlertid ha noe bedre dokumentasjon hva fremdriften angår.

Automatiserte tester. Det er satt opp noen automatiserte tester, og disse ligger på det nivået vi forventer i prosjekter på Drupal 7 i 2012. Drupal-milijøet generelt har en vei å gå i å innarbeide testdrevet utvikling.

Tilgangskontroll. De rollene og rettighetene som er satt opp, tjener en enkel modell for tilgangskontroll, noe vi støtter. Det er klassiske velprøvde Drupalløsninger som ligger til grunn, og disse besørger også et stabilt nivå med høy sikkerhet.

Sikkerhet. Ved gjennomgang av sitebuild, custom kode og tredjepartsmoduler har vi ikke gjort bekymringsverdige funn når det kommer til sikkerhet. Vi har minimal tilgang til informasjon om driftsmiljøet, men dersom servermiljøet settes opp etter best practices og contrib moduler er oppdaterte ved lansering, er sikkerheten godt ivaretatt.

Konklusjon. Vi mener UiB har gjort et klokt valg i å velge Drupal som plattform for sitt nettsted, og anser det som sannsynlig at prosjektet vil bli en suksesshistorie for både institusjonen og plattformen. Vi tror videre at UiB vil ha nytte av å pleie kontakten med Drupal-milijøet nasjonalt og internasjonalt i tiden som kommer, og at det er viktig også i kommende faser at teamet får tid til å holde seg oppdatert på trender i miljøet, slik de har gjort til i dag.

Development setup

We've done successful attempts at setting up the site for local development. The repo at drupal/ points to a repository at uib.no, this was changed to git@github.com:uib/drupal.git

Imports runs ok using the VPN, following the instructions provided in git. However, when doing larger imports using bin/site-drush, we experienced occasional issues with the PHP memory limit. Using normal drush outside site rep seemed to work normally.

We have also used the test site at w3.uib.no as an example site.

General sitebuilding

Most of the sitebuilding is stored in code, exported as Features. Content types and fields, and even settings using Strongarm module. This is a good thing.

Content types

The generic configuration on every content type is properly configured, with the appropriate settings on key areas, namely: form settings, publishing options, revisions and multilingual support.

The current content type descriptions gives the right info to the user about the content types purpose. However, we suggest that these descriptions follow a standard format - 1 short line sentence with the info [what is] + [what it's used for] - and are also translated to Norwegian (we assume the latter is planned).

Fields

The fields configurations follow a strict and appropriate naming convention: short, descriptive and predictable machines names, with labels names that are descriptive and independent from machines names, when appropriate.

The fields that are for admin/development use only, should be identifiable as such. For future development, also mention if its removal is possible on a later development phase.

The field types chosen for all fields are the most appropriate, given the solutions available in Drupal, and the widgets in use for each field provides the best user experience for content editing.

A term reference field type would be a better option for "Type" in Article content type, and probably in Area and Study also. The reason is that Taxonomies provides more and better options to handle related content and features - such as long descriptions, images, translations, views - than a List field.

Most of the fields have an associated descriptive help text, but some fields are lacking this help text. Also, the help text copywriting style isn't consistent: some are actions, others are instructions or descriptions. We suggest a review of the fields help texts, to fill the missing

descriptions and standardize on a format, for example: [what is this field] + [what it is used for] + [options/special instructions].

The content editing panels have all fields correctly organized, with the field group properly set up.

Displays

Currently, only the Default and Teaser view modes are active, with fields display settings correctly set up on Default view mode. Given the actual display requirements for the platform, these are the only necessary display features; the additional view modes and field settings provided by solutions such as Display Suite are not required at this phase.

Although, if the content display requirements increase substantially in the future, UiB should consider the use of Display Suite module, for a better maintainability, performance and development speed.

Users

Overall, the user generic settings are correctly set up. The e-mails messages templates might require some customization.

The user fields follow the same strict naming conventions as the content types, and have the appropriate types and widgets. It could use field groups to organize these fields on the editing panel, similar to the content types.

The permissions, an important and often neglected area, seems to be correctly setup for all user levels (roles).

The users are assigned to roles defined by levels of permissions, opposed to the traditional semantic roles common on Drupal solutions. This seems to be a more appropriate solution for the new UiB platform, given that the complexity of user related requirements could lead to duplicated and unmaintainable user permissions, if the traditional approach was used.

Panels & Blocks

The dashboards for the Webdesk section are built using the Page Manager from Panels, which is the right solution to built this kind of feature. Given its almost unlimited integration with other platform components, it can contain and integrate almost every context and content available on the system.

The Drupal core block management system is replaced by the Context module suite, a good decision that will allow the platform to scale without making it unmaintainable. Additionally, this solution provides more contexts for more granular and precise control.

Although the Panels and the Context modules tend to overlap/duplicate functionality, their current use on the platform is the most appropriate: Panels is currently being used for content and application building, while Context is used for global layout and structure control.

Views

The Views currently set on the platform are in development, with some on early stage of development. Analyzing the already finished Views, we are sure the developers have a deep understanding of Views features and how to put them to good use in the platform.

Views are used on the platform to generate the necessary pages and content blocks for frontend users, and backend functionality for more privileged users. The Views Bulk Operations module is also used to provide additional functionality and UI improvements on platform administration.

Views settings are reused through pane displays when appropriate, and its use reveals a good knowledge of the different panes types that are available, their features and when it's appropriate to use each of them.

The right formats were chosen and correctly setup. In some cases, on non-admin Views, we recommend to use the Style settings, to output a more lean and semantic markup.

Regular filters are correctly set, and contain additional checks that prevent the Views from outputting empty and/or bogus results. Additionally, contextual filters are correctly used and related to Relationship options.

Overall, we didn't find any relevant issues with the Views currently in use. There are some small improvements that could help its future development and maintenance:

- A few machine names are correctly set on Views panes, using a short, descriptive and unique name; we recommend doing this for all panes, to ease the module/theming development that involves these panes.
- Some views are missing descriptions and tags - we recommend filling these in to help future developers.
- Some views don't seem to be in use? If they're not in use, we suggest that they're disabled or removed.

Menus

The menu setup workflow follows the right sequence: menu items are added from the linked content/section itself, and then reorganized using the menu management UI.

The multilingual menus are provided by creating additional menus for each language, instead of using menu built-in i18n support. The menu built-in i18n support on Drupal menus has some quirks, and can be hard to manage, especially for non-tech savvy users, so this duplicate menu option seems appropriate.

Tabs/local tasks are used when appropriate, and are correctly set up.

i18n

There's a set of i18n modules available for use on the platform. It seems that the i18n is in an early implementation phase, with the initial options correctly setup.

The default language is “English”, the right option to simplify the string localization workflow. The language detection and selection is based on URL and a set of custom UiB rules that maps URLs to language; we suggest that “Follow the user’s language preference” is also added.

The content published on the platform already have the right language options assigned, with language neutral for content that can be shared between languages.

Contrib modules

The project is, at the time of writing, using 60 contributed Drupal modules. This amount is considered normal for a site with this complexity.

Going through the list we find that most of the modules in this project are in good shape with a stable releases, and we have positive experience using them in our previous projects. The module Bot, we have no direct experience with, although we consider it rock solid since it has been part of Drupal.org for many years.

There are, however, some beta releases in the repository, which needs to be addressed. Those identified are:

- Feeds Xpathparser - Beta
- Field Collection - Beta
- Fieldable Entity - Unstable
- Ldap - Beta

For modules considered unstable, we recommend contacting the maintainer, and provide testing and patching resources needed for releasing a stable version.

Code review - Custom code (PHP)

Custom code is in this project added directly to the features’ .module files. This is ok, however it removes some of the general overview of the custom code used in the project. Features code are always dependent on the contrib modules that exported them, and can therefore be considered a ‘middle ground’ between custom and contrib code. So consider having any custom code separated from the feature modules, together with good description of what the code does. The standard for module separation is generally considered to be Features - Contrib - Custom (or: name of project), in it’s own subdirectory of sites/*/modules directory.

Coding standards

The Coder module is important to use on a regular basis, to identify bad code and ensuring all code is in accordance with Drupal’s coding standards. 95% of the 50+ currently reported issues are loop spacing and commenting issues, there is only one critical element. We recommend using the code review on a regular basis, in addition to adjusting each developer’s IDE in compliance with the coding standards. In doing so, the project maintains a good code profile in compliance to standards.

We suggest that some improvements be made, for better code legibility and adhere to

conventions that future Drupal developers would expect.

- Avoid using inline control structures - they're not common in Drupal PHP code, and decrease legibility when mixed with curly brackets
- The use of isset() is preferable over empty(), when verifying the existence of a variable for comparison, as it avoids unnecessary error notices.
- Inline comments should use "// Comment" syntax, instead of doc block syntax.
- When using Drupal's predefined constants, it is generally considered a good practice to use these instead of their defined values. Replacing values with constants eases eventual upgrades (where constants change), and readability.
- Instead of querying arg(0+1) for current node, you can use menu_get_object() to retrieve the current object in the menu trail. This will return false if no node.
- Do not use 'e' modifier in preg_replace(). Instead, use preg_replace_callback().
- t() should not include spaces in the beginning or the end of the value string.

Other suggestions

Here are some more generic suggestions from the code review:

- When hiding fields for use only by programmatic sequences, it's generally a good idea to keep them available to the node form. If needed, they can be exposed without finding and changing custom code. Consider using hidden field formatter modules for hiding them.
- Do not query the field revisions table when querying for field results. The revision table can hold multiple instances of a node, and it is therefore discouraged to use this for fetching field data, other than in very special cases. Consider using field_data_* tables instead.
- If deployment on production is not to include devel, do a sweep of the repo's code now and then, and eradicate dpm()'s etc. Keep the test code either out of repo altogether, or comment it out and document it for later use.
- Languages are not in use on any fields saved to features, so when making queries for node data, you may consider wrapping with entity_metadata_wrapper(), part of Entity API, when manipulating entities through relations. Using this makes traversing a whole lot easier.
- You block outgoing mails completely on development. To get an overview of what mails are actually sent out, you can consider setting up Reroute Email as part of the development environment.

Code review - Frontend code (js/CSS)

The Zen theme is the most appropriate base theme for the new platform, considering the specifications regarding the presentation layer, specifically those related to flexibility and extensibility of the templates.

Besides some modern features, like full HTML5 support, complete SASS/Compass support and a responsive design framework, Zen theme provides Drush support, which can be put to good use on the creation workflow for new sections and sites on the UiB platform.

ui_zen theme

The UiB sub theme seems to follow the theming conventions of Drupal in general, and Zen theme in particular:

- PHP logic is placed on template.PHP
- Tpl files are logic less and almost exclusively only output markup + variables
- CSS follows the organization imposed by Zen theme

PHP code

Some improvements are suggested for improving the template and theming code for compliance with coding standards and best practice:

1. The use of isset() is preferable over empty(), when verifying the existence of a variable for comparison, as it avoids unnecessary error notices. Example: on template.PHP/uib_zen_preprocess_html(), it should check if $user->field_grid['und'][0]['value'] is set and then compare it.
2. Variables should be set only when necessary. Example: on template.PHP/uib_zen_preprocess_page(), $variables['page_title'] and $variables['page_title_link'] are set 2 times on some cases.
3. Comparisons should only be performed when necessary. Example: on template.PHP/uib_zen_preprocess_node(), some of the comparisons on line 196-218 could be avoided using a switch() statement instead of if() statement.
4. When possible, all PHP logic should go into template.PHP. Example: on templates/views-view-table--uib-webdesk--page.tpl.PHP, the logic for the $user_role variable should be done in template.PHP, into a hook_preprocess function for that view.

SASS/CSS code

The Zen theme used provides a structure of CSS files that meets the way Drupal content is generated and outputted. This structure doesn't fully meet new CSS coding best practices, such as SMACSS and OOCSS, but isn't totally incompatible with those principles. In fact, Zen 7.x-5.2 will include some SMACSS concepts (At the time of this writing, UiB is using 7.x-5.1).

We strongly recommend that the theme adopts SMACSS and OOCSS principles when possible and appropriate, for the reasons we mention next. Given some field options, it seems that this is partially planned, but the current CSS doesn't follow those concepts.

This is important, given the specifications for the new platform, which needs a good theme foundation to build upon, that should excels in these aspects: flexibility, scalability and maintainability. Since Zen 7.x-5.2 already implements the SMACSS concepts, implementing SMACSS on the current uib_zen will ease the future update of the base theme.

SMACSS

The main concept of SMACSS is to categorize CSS into base categories, using loosely defined naming conventions/rules. This allows to write less CSS, with less CSS specificity, that is more easily overridable and/or replaceable.

The current CSS organization has some issues on these aspects:

- CSS specificity is on some cases higher than it needs to be, with makes it more difficult to override on sub-themes/variations.
- There's some repetitive properties that probably could be avoided, if relying on naming

conventions / classes.

- File organization can be ambiguous, even considering the Drupal conventions. Example: views-styles.CSS holds the views styling, but a view can be outputted as a page, block, panel, etc. If a sub-theme wants to customize the visual of blocks, it needs to keep track of the definitions and change it different places.

OOCSS

The OOCSS approach consists mainly in two principles:

- separate structure and skin
- separate container and content

Since this is the foundation theme, it needs to control the structure, and consequently the containers, to provide a consistent structure to all variants and applications of the theme. Then it should decouple the skin and content, which will change depending on the context, making it a option and easily customizable for implementations built upon uib_zen theme.

Example: the menus have different variants (dropdown, collapsed etc.) but share the same visual, and could possibly have a different visual on a different section. Separating the visual properties from the structural properties would allow to share that visual between menu variants, and consequently use less CSS, and facilitate the visual changes on sub themes variations.

Global SASS/CSS best practices

These are suggestions for some small improvements, that are generally accepted as best practices and can improve code maintainability and quality.

- Selectors should follow the Zen convention - short name with inner 1-line descriptive comment, and using the class selector when possible.
- Drupal CSS coding standards should be applied http://drupal.org/node/302199 - in some cases the code is slightly hard to read, and this could possibly result in inconsistency and repetition.
- SASS code nesting should be used more often, to avoid duplicate selectors and make code more legible.
- Compass helpers should be used for CSS3/x-browser properties support (border radius, gradients, box shadow, opacity, etc) - the helpers are generally very proven and allows to write less code.
- Use SASS variables for shared visual properties such as colors and font families.
- Avoid the use of !important for visual properties.
- Decrease CSS specificity, making use of classes and disallowing overqualified elements when possible.

Possible bugs

- On template.PHP, line 136 - it’s possible that this condition returns true, no matter what value is/is not set.
- On views-view-table--uib-webdesk--page.tpl.PHP, lines 43, 62 - it is possible that this comparison returns true, if some hook insert another class in $header_classes.

Backend performance

All the time we have spent with the code, we have not identified any bottlenecks that cripples performance. Typical pits would be colliding modules, bad queries or simply too many modules.

The memcache module is not present in this installation; we take that as a sign that the installation is not yet tested with Memcache. The final server setup itself is partly documented on Redmine: A Nginx setup using varnish and balancers. We support these decisions.

Other than that, we recommend using Entity Cache http://drupal.org/project/entitycache for better caching of server code.

Performance analysis

We've analysed page request using Xhprof and New Relic, using an import with approximately half of the available import data. The tests show a traditional resource usage:

- Lower on admin side, but also relatively low on frontend
- Memory peak for normal page requests are at 28-30MB, which is very effective.
- Node and views are the most time consuming modules.

For data generation, we used a selenium test clicking random site links for 30 mins. This revealed a list of what modules uses the most of the resources. The test was performed with all cache off.

Figure 1: Load during selenium test for anonymous user

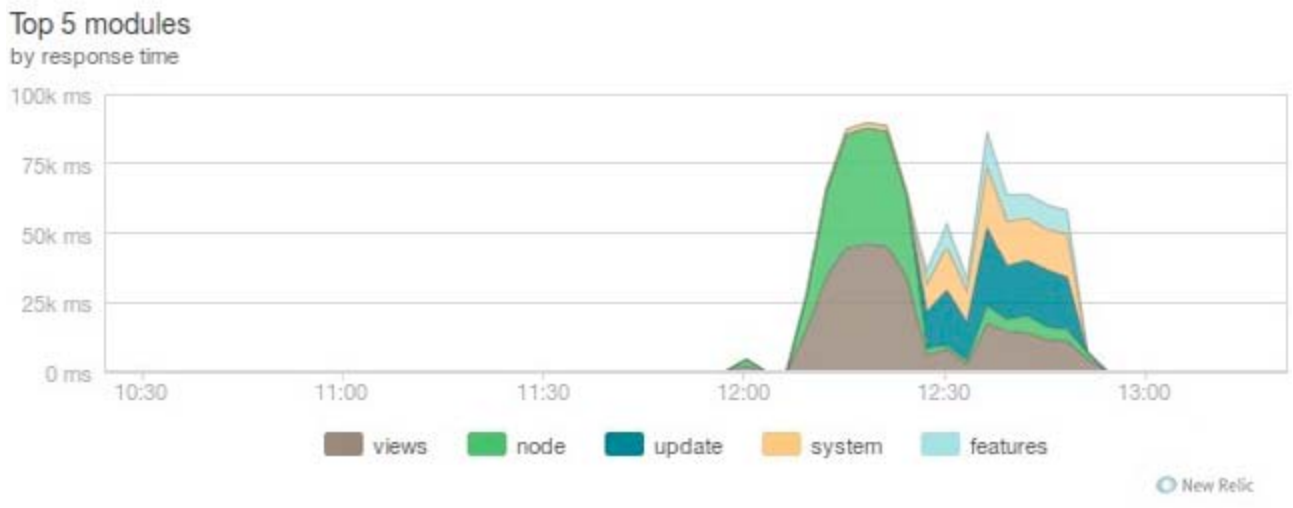

Figure 2: First 30 minutes: test as anonymous user Last 30 minutes: test of administration pages as logged-in administrator.

Most time consuming parts:

- Views – The traditional top consumer, since it does the most queries on landing pages etc. All frontend views should be cached at some level.
- Node – In our test environment, this is due to the fact that Entity Cache and core cahes are off. It is likely to decrease drastically in a production environment, even as logged in.
- i18n_field is a time consuming module due to the fact that it intercepts field strings and translates them. This is especially heavy during node view operations, but is normally cached during pageviews.
- Menu Block

From here you can see where caching efforts should be put in. Panels, which is generally a very consuming module falls far down on the list, due to the fact that it has few implementations in the project. So the top priority for caching is Views and nodes.

Frontend performance

The base theme used on the platform outputs a lean markup, and introduces some of the best practices regarding CSS performance and HTML5 support. The customized sub-theme seem to follow these practices and, at this stage of development, didn't introduce any bloat in the markup language and kept custom tpl.PHP files to a minimum.

Just by activating the Drupal performance mechanisms for page cache and CSS/JSS aggregation, the site gets a Grade A on Yahoo!'s YSlow performance tool, and a Google's Page Speed Score 100.

It's expected that this grades decrease slightly, as more content and features are added to pages, but there's still some room for improvements, regarding CSS optimization (see 'Code review - Frontend code') and server configurations (compression, etags), that should keep the high performance standard of the front-end code.

The content that is displayed using Drupal fields introduces some non-semantic markup bloat in most cases. We recommend the use of the module Fences, to output more lean and semantic markup; the module is maintained by the same author of the Zen theme, so there should be no conflicts between those.

The Drupal performance settings are off and there's no information regarding the server environment that affects front-end performance. We assume that this is because the project is in the development phase, and that once on the production, the optimization configurations will be set.

The media settings seems appropriate.

Additionally, it should be considered to lazy load images in content heavy pages (galleries, publications etc.) to improve the loading and prevent unnecessary assets loading, saving bandwidth. Drupal already has contributed modules that implement these features.

Git workflow and history

Submodules

UiB is using git submodules, which makes sense in this kind of project. Having contributed modules and core separated in a submodule is recommended. It enables you to use a certain branch of the Drupal tree, while not being linked to any contrib code when maintaining the custom code. Commit history is then separated between custom and contrib code as well, which is great for long-term code maintenance.

Commit messages

The project's commit history is preserved with issue numbers modeling the standards used in the Linux kernel. There are also project guidelines documented on how to avoid generic merge messages.

The result of this is a currently clean and effective commit history. English commit messages are favorable for potential external developers/reviewers, as well as code comments.

It is also a best practice to link to complete commits from the issue queue at https://rts.uib.no/projects/w3, like you already do before closing issues. For ease of readability of the issue queue, you can consider linking directly to the commit online, either on github or other service, or, if possible, use a formatter on rts. This makes it easier for those without git setup to review changes on the fly and thus eventually deploy code review routines.

Deploy system

We find no documents lining out how the development process should go forward once the site is in production. We recommend allowing separate branches for experimental development, but with strict conventions, thus not allowing branches to "float out".

When the site has entered into a production phase, you can optionally start using git flow as a branching model (https://github.com/nvie/gitflow). As multiple subprojects and modules are in development by multiple developers, one common master branch may not be the best workflow. Git flow's commands uses the master as the main branch, and allows you to easily branch out using a standardized pattern.

We also recommend looking into more possibilities:

- Vagrant development environments allows for easy deployment of test sites, allowing all developers to have the same development environment as exist on production. It allows for building a complete, structured test environment once, and allow every developer to use the exact same setup. Vagrant is well integrated with Drupal using modules and Drush extensions.
- Puppet
- Ballista – Git deploy control system https://github.com/allerinternett/ballista

Documentation states that Jenkins will be part of the development workflow.

Timeline

We have looked at the timeline for release provided by UiB, and then reviewed the issues in Redmine; this gives us a feeling of the time/work ratio. Conclusion: The release plan looks to be well thought out, and has the needed buffer for potential changes. The amount of high priority bugs and new features should not be a reason for concern for the internal team, provided that they don't take on other tasks during development phase. We would however recommend the team to make a bold list of priorities even at this stage, in case parts of the team should not be able to work all allocated hours.

For the first Dec 14 release, the issue queue needs a cleanup (http://goo.gl/242IZ). We suspect that some features planned for release will either not make it in, or are mostly finished. In the latter case, the project would benefit from better documented issues in Redmine, as this would make team changes and stressful releases less painful. Because of lack of documentation, progress on some medium size issues are unclear. Example: https://rts.uib.no/issues/154. Some other tasks looks not to be finished in time, but they are not blockers.

We later learned about the hospitalization of a key team member; hopefully he will recover well. We advise that all of his tasks are formally re-assigned in Redmine.

Project management tools

We wanted to make sure the team was using a centralized solution where both bigger features and small bugs were connected with the release plan and assigned to real persons. The team does indeed do this, and they seem to do it it a systematic way. They use the open source tool Redmine for this, which we believe should serve them well, even for further development.

Automated testing

The documentation mention that Selenium tests will run at float.uib.no/jenkins whenever something is pushed to the repository. Together Jenkins and Selenium are a well-documented approach for running automated test environments.

Having automated environment triggered whenever the code is pushed, ensures that the test workflow is always followed. Its importance in the production workflow cannot be overestimated. It's a plus if the patch submitter receives a report of the test once it's done as well.

The planned approach is good and sustainable. We were not able to do practical tests of the current workflow in its current state. However, the structure of the test code looks good.

There are numerous language alternatives when setting up headless selenium tests, and the python approach in this project is both maintainable and expandable.

Access control and permissions

The permission structure is divided into 3 different role levels in addition to the regular authenticated role:

- Level 1 - Lowest
- Level 2
- Level 3 - Highest

This maintains a simple, maintainable permission system. If the access permission structure will be hierarchical, this is a good approach. It can eventually be supplemented by independent roles outside the hierarchy, or with additional top levels.

Security

The project's custom code is fragmented together with features. We consider Features static code which seldom interfere with other code in the way custom code may do (like poorly written hook_form_alters, new modules, updates etc). These should be easily identified.

Contrib modules in beta is a constant security issue, and should be addressed with resources towards the community, and fixing such issues. Many tend to think that "as long as it's stable for me, I can use it". However, security issues appear in unstable & beta contribs, and often they affect sites where it's stable as well. In this project's current scope, this issue centers around the 4 modules in use by the project currently marked unstable. Pushing resources towards them with bug fixes and development time is a great way to increment the project's security level.

With current level of custom code, the approach with mixed features and custom code is maintainable, but as mentioned before, we recommend separating custom code and features into separate structures.

Documentation

The code repository provides full documentation for coding standards used in the project, platform specifications and additional guides to help future developers. The documentation is generated by Markdown and Doxygen, two widely used and proven solutions.

During our analysis, we quickly got the scope of the project and successfully set up local developments environments, following the documentation guides, which proves the good quality of the documentation.

The project code is properly documented, following Doxygen formatting conventions, which allows developers to quickly extract and access the code documentation.

Reviewed by: Alf Harald Sælevik, Roberto Ornelas, Henrik Akselsen and Per André Rønsen