Dryad System Curation Proposal: A Technical Review

prepared for Dryad Project by Akio Sone
November 2009.
version 1.1.

Introduction

Aims of this report

The aims of this report are, as requested, to review the Dryad System Curation Proposal[1] (hereafter shortened as "Curation Proposal") and analyze its technical implications.

Structure of this report

This reports consists of five sections including this introductory section including one appendix. The second section summarizes the Curation Proposal in terms of new features to be added to Dryad. The third section first enumerates important constraints and considerations before the requested curatorial features are evaluated and then each requested feature is assessed by its benefits and costs. The fourth section concludes this report and an appendix contains technically oriented material about curation tools.

Definitions

This report defines key terms as follows:

Stakeholders

The Curation Proposal lists two stakeholders: curation staff and Dryad end users. This report assumes that the above Dryad end user includes not only depositors/submitters but also future users of Dryad who visit Dryad site and browse and download an article and/or data files from it.

Dryad

The current Dryad submission component and does not includes auxiliary components to be added unless otherwise stated.

Base assumptions about coding estimates

Coding-related estimation figures are based on the following assumptions:

1. A full-time developer who is hired by Dryad project will carry out development tasks.
2. GUI-related work packages would employ an iterative development approach and there would be at least one feedback/GUI-modification cycle.
3. Development time figures includes time spent for not only development itself but also collective decision-making about specifications and business rules.
4. A week means 7 working days, not a conventional week (5 working days + 2 holidays)

The Curation Proposal: An Overview

Features requested by the Curation Proposal

The Curation Proposal is comprised of three major parts: (1) the first part reviews Dryad submission system in term of the current and proposed deposition procedure and desirable features/tools to be added, (2) the second part enumerates the system requirements of the future Dryad, and (3) the third part shows GUI-mockups according to the requirements detailed in the previous two parts. This report focuses mainly on the second and third parts

[1] Carrier, Sarah. 2009. "Dryad System Curation Proposal." (https://www.nescent.org/wg/dryad/images/b/b0/CurationProposal.pdf)

except for design ideas on Dryad because there are overlapping between the first two parts concerning desirable features and descriptions about the deposition process are out-of-dated as of the November of 2009.

Major design considerations

As a frame of reference, it must be beneficial to recap key design considerations for Dryad mentioned in the Curation Proposal (page 1). These key design considerations for the future Dryad are: (1) ease of metadata generation (for researchers) and deposition (for depositors); (2) curatorial functionalities that preserve metadata and guarantee the permanence and usability of submitted data. These design considerations are later used as criteria for assessing the benefits of each of the requested curatorial feature.

Requested Curatorial features for the future Dryad

The Curation Proposal enumerates system requirements for Dryad's to-be-developed curatorial component in eight major categories. This report re-organized these system requirements along with desirable curatorial features described in earlier sections into 15 features; Table 1 lists these 15 features.

Table 1 Curatorial features requested in the Curation Proposal

feature Number	proposal page	Feature (Item numbers are taken from System Requirements Section)
1	13	external spell checking
2	14-16	controlled vocabulary server including 7. auto-suggest (with real-time word-updating) functionality similar to CONTENTdm
3	14	text-extractor from PDFs
4	14	1. contact methods from the depositor to curators
5	15, 4-8	2. simpler method to add data files to a publication later
6	15	3. reference/inheritance-based intelligent data-entry/editing
7	15	4. DOI access via ISI or PubMed
8	15	5. GUI customization according to a curator's skill or qualification level
9	15	6a. tray mechanism (untouched items)
10	15/21	6b. batch-processing
11	16	6c. processing-status reporting/ monitoring
12	16/10-13	6d. integrated curation tools
13	16	6e. search/report of data files by profile/usage-level
14	16	8a. version control: metadata
15	17	8b. version control: data file

These 15 curatorial features will be individually evaluated in detail in the next section.

Curation tools and curatorial GUIs

The Curation proposal presents GUI mockups in its third segment. From the viewpoint of a repository developer, a few comments about these mockups are provided here.
First, for a certain functionality such as file-format identification for uploading an individual data file (not bulk processing), curatorial tools may not always need a curator's intervention or decision via GUI; default decision-making can be hard-coded and ordinary cases are automatically handled by the backend of Dryad, and a curator's intervention via GUI can be limited to special-attention-required cases only. Thus, when curatorial tools are integrated into Dryad, it would beneficial to separate curatorial functionalities that do not need a curator's intervention via GUI from those that always require a curator's decision.
Second, DSpace community now has StatisticsAddOn[2] and it would be worthwhile to test whether StatisticsAddOn could be used for Dryad's usage-related monitoring/reporting functionalities. If this addon satisfies Dryad's requirements, designing Dryad's report-related pages would be based on the framework of StatisticsAddOn, especially visual elements such as charts.

[2] http://wiki.dspace.org/index.php/StatisticsAddOn

Evaluation of the 15 Requested Curatorial Features

Before each of the requested curatorial features in the Curation Proposal is evaluated, key constraints and other important considerations for their evaluation are outlined below. These constraints and key considerations later guide the prioritization of features to be developed by Dryad project.

Key constraints on Dryad and other considerations

1. New version(s) of DSpace
The most significant constraint must be how DSpace, which Dryad is based on, would evolve in the near future and add new features. Because if its soon-to-be-added features include one mentioned in the Curation Proposal, it would be wiser not to hastily choose an in-house development option but to wait/see features developed by DSpace project/contributors and divert Dryad's resources to other higher priority items. The soon-to-be added batch editing functionality (DSpace release 1.6.0)[3] is this example.

2. Schedule integration of the StandAloneSubmission component into DSpace by @mire
The next significant constraint is how the current StandAloneSubmission component is integrated into DSpace by the @mire team. Some of the features in the Curation Proposal may be affected by this integration process, either not at all or some degree (slight or considerable). Given this future integration, it is prudent to start in-house development items less related to the integration and postpone those that have side-effects on it.

3. Interaction with the persistence layer
Generally speaking, a feature that does not call or require the persistence (database) layer of DSpace could be relatively easier to implement as a separate, standalone coding project; for example, a GUI-development-required feature that does not interact with the database or other base-layers could be this kind. On the other hand, if a feature that relies on the database in terms of its functionality and there is no DSpace addon that could be adopted for this feature, implementing this feature in house would need longer development time.

4. New-business-rule(s)-to-be-established
If a feature necessitates information about not-yet-established business rules with which potential stakeholder must agree, such a feature would require a lead time for the agreement before a developer actually starts to work on an in-house implementation project. Also, a feature that introduces new business rules tend to require changes to the persistence layer, and these changes result in a longer implementation time.

5. Components/modules developed by others: HIVE Project
Among the aforementioned 15 requested features, there are a few features to be developed by another Dryad sub-project (HIVE project). In terms of dependency, the development of these features can be independently carried out until these features are fully developed and pluggable into Dryad itself with certain pre-arranged protocols, i.e., neither of these features nor Dryad depends on one another. Therefore, this report focuses on their benefits on Dryad and does not discuss their development in detail.

6. GUI-development involvement
If a software project requires to develop a GUI (graphical user interface) for its end-users, in terms of usability, it is crucial to have a test run by end-users and get their feedback, and then tweak the original GUI according to their feedback. If their feedback is extensive, an additional, final test run by end-users would be necessary. These iterative development stages would require longer development time compared with the development of a back-end module that does not have direct interactions with end-users through GUIs.

[3] http://wiki.dspace.org/index.php/DSpace_Release_1.6.0_Notes

7. Relationships among requested features

While the 15 requested features are diverse, several of them depend on one anther. If these features rely on a common, base functionality, it is important to factor out it and implement it first. For example, the tray GUI feature (ninth), the batch processing (10th), and monitoring tool (11th) assume that a stored item's relevant status data are saved on the database and these data are then utilized for later curatorial functions such as reporting and batch processing. Also, this dependency requires a certain agreement on relevant business rules among the Dryad curator and Dryad-user community before or during the implementation period.

Table 2 tabulates relationships between the 15 requested features and key constraints and considerations. Except for the third column (DSpace own development available), a requested feature that does not have many check marks would be easier to be implemented (the ease of implantation does not mean more benefits, of course).

Table 2 Requested features and constraints on them

Feature Number	Requested Feature	Developed by DSpace	Integration into DSpace	Interaction with persistence layer	New business rules required	Developed by Others	GUI-development involved	Relations among features
1	external spell-checking						✔	
2	controlled vocabulary server/ auto-suggest with real-time word updating			✔		✔	✔	
3	text-extractor from PDFs					✔	✔	
4	contact methods from the depositor to curators							
5	simpler method to add data files to a publication later		✔			✔		
6	reference/inheritance-based intelligent data-entry/editing		✔			✔	✔	
7	DOI access via ISI or PubMed						✔	
8	GUI customization						✔	
9	tray mechanism			✔	✔		✔	✔
10	batch-processing	✔ (1.6.0)		✔	✔		✔	✔
11	processing-status report/monitoring			✔	✔		✔	✔
12	integrated curation tools			✔	✔		✔	✔
13	search/report of data files by profile/usage-level			✔	✔		✔	✔
14	version control: metadata			✔	✔	✔	✔	
15	version control: data file			✔	✔	✔	✔	

Table 2 was later utilized to assess each feature's complexity when its benefits and costs were estimated.

Requested features and their benefits and costs

After having enumerated key constraints and considerations, this report employed the following set of questions to characterize each of the 15 requested features. These questions are as follows:

(1) code change: type
(2) code-change: impact
(3) time-required
(4)immediate benefits
(5) indirect benefits

With these questions, the 15 features were assessed and their results are summarized in Table 3. In terms of code change (type), most of the 15 requested features need to add a newly developed (in-house) module or third-party one to Dryad. The impact of code change would largely depend on whether such an addition of a module may require extensive GUI-make-over including new web pages (more coding/testing time) or new columns or tables on the database with their accompanying backend programming (more coding time). Immediate benefits are typically more efficiency(time-saving) and/or higher quality, and indirect benefits are various from higher quality metadata to higher publicity of Dryad.

As mentioned before, for those features to be developed by HIVE project, code change and required time columns were not evaluated. For HIVE project, the requested features 2 and 3 would be in-house development with assembling third-party modules whereas for Dryad side, if these requested features (2 and 3) are ready to be integrated, required integration work would be (1) to hook up these features into the backend of Dryad and (2) to modify related web pages of Dryad. Of course, before this integration, there must be an agreement about the protocols and interface-related specifications between these features and Dryad.

After having evaluated each of the 15 requested features, this report looked into relationships among these features in terms of benefits and implementation time; results are shown in Table 4. When this report combined similar or duplicated features in the Curation Proposal into a single feature and boiled them down the 15 features, the granularity of each requested feature was kept intact. Consequently, their granularity considerably varies form feature to feature: many of them are narrowly defined and can be turned into work-packages immediately, but there are a few features that can be further divided into sub-features or factored out as mentioned above. Thus, when Table 4 is interpreted, this granularity issue must be taken into account. Clearly there are no high-benefit-and-easily-implementable requested features (north-west corner in Table 4).. The dispersion of requested features on Table 4 reflects the above granularity issue: more time-consuming but more beneficial features are coarser and less time-consuming and beneficial ones are finer.

An Implementation Plan

As motioned earlier, Table 4 tells there is no requested feature that is both shorter implementation time and more benefits simultaneously. A second-best plan how to implement these features would be (1) to finish shorter-lesser beneficial features (4, 7, 8) first; (2) to re-structure reporting/monitoring related features (9, 11, 13) according to their dependency on the persistence layer (database) and implement them. As for the version-control feature (14 and 15), it would be wise to wait and see how open-source software development projects, especially, those related to a content management system (CMS) and digital archives/libraries, would deal with this functionality for a while. If versioning functionality implemented by an existing open-source CMS such as Apache Jackrabbit fails to meet Dryad's requirements, the in-house development of versioning would be seriously considered; otherwise, customizing an existing versioning module according to Dryad's needs may be enough.

Table 3 Requested Features: their benefits and costs

Number	Feature	handled by	code change type	code-change impact	time-required	immediate benefits	indirect benefits
1	external spell checking	Dryad	add (use existing packages)	narrow (GUI)	7 days	time-saving	higher-quality metadata
2	controlled vocabulary server/ auto-suggest with real-time word updating	HIVE				time-saving	more consistent metadata
3	text-extractor from PDFs	HIVE				time-saving	more metadata available for search, etc.
4	contact methods from the depositor to curators	Dryad	add: in-house development	narrow	3 days	mis-communication-prevention, time-saving	higher-quality metadata
5	simpler method to add data files to a publication later[4]	@mire			handled by @mire?	time-saving (ease of use)	more uploading of data files
6	reference/ inheritance-based intelligent data-entry/editing	Dryad (+HIVE)[5]	add: backend module and GUI modification	narrow(DB), time-consuming (GUI)	3 weeks	time-saving (no more repeating cut/paste)	higher-quality metadata
7	DOI access via ISI or PubMed	Dryad	add: use existing packages and in-house development	narrow (GUI)	5 days	time-saving (no waiting time)	shorter lag-time for on-line visibility
8	GUI customization according to a curator's skill or qualification level	Dryad	add: in-house development	narrow	5 days	less intimidating for new users	wider adoption of Dryad system
9	tray mechanism	Dryad	add: new feature	wide (GUI+DB)	4 weeks	time-saving/reminder	easier workflow planning
10	batch-processing	DSpace/ Dryad[6]	add: new feature	narrow (DB+GUI)	4 weeks for the 1st function	time-saving	more consistent metadata
11	processing-status reporting/ monitoring	Dryad	add: new feature	narrow (DB), time-consuming (GUI)	4 weeks	time-saving/reminder	easier workflow planning
12	integrated curation tools[7]	Dryad	add: integration (use existing packages)	depends on each tool's pluggability	2 weeks per tool	time-saving and long-term preservation	more metadata available for search, etc. higher quality metadata
13	search/report of data files by profile/usage-level	Dryad	add: in-house development + DSpace StatisticsAddOn	narrow (DB) , time-consuming (GUI)	2 weeks	time-saving (search)	better public relationship/ more media-exposure | visibility of Dryad
14	version control: metadata	Dryad	add a third-party module or in-house development or both	narrow	5 weeks (3rd-party module is used)[8]	more up-to-date provision of metadata	easier tracking of update history
15	version control: data file	Dryad	add a third-party module or in-house development or both	narrow	5 weeks (3rd-party module is used)[9]	more up-to-date provision of data files;	easier tracking of update history; help replication efforts

Note: DB stands for a database.

[4] Here assumes that the forthcoming integration of StandAloneSubmit Component into DSpace by @mire would cover this requested feature.
[5] The assumption here is that the backend module would be developed by HIVE project and Dryad would use the backend module.
[6] Partially covered (bulk import functionality would be included in rel 1.6); other batch-processing functionalities are not covered by this new feature.
[7] A relatively new omnibus curation tool, File Information Tool Set (FITS), includes many format-identification/metadata-extraction tools (see Appendix)
[8] This estimate assumes a third-party versioning module is adopted without extensive modifications.
[9] ditto

Table 4 Requested features: their overall benefit and development time

	← Shorter Time	Longer Time →
More Benefits ↑	8 Customizable GUI 7 DOI Access via ISI 1 Spell Checking	12 Integrated curation tool 9 Tray system 11 Status-report/monitoring 6 intelligent data-entry 14/5 Version control 10 batch processing
↓ Less Benefits	4. Contact Methods	13 Data-file usage report

Note: Features to be implemented by non-Dryad project were excluded from the table;
The implementation time of the feature 12 is per tool.

Summary

This report has summarized the Curatorial Proposal concerning the 15 selected curatorial features and assessed their benefits and costs. While the set of these 15 features does not present a linear, anyone-can-agree-with course of development steps for a developer, this report has at least revealed relative relationships among these 15 features in terms of their immediate and indirect benefits and required implementation time. These relationships among the requested features could provide an actionable guide, such as which feature should be implemented ahead of others, for Dryad's developers.

Appendix

This appendix contains technically-oriented materials that were excluded in the main body of this report.

How to integrate a curation tool into Dryad

Dryad is a Java-based web application; thus it is natural to assume that if a curation tool is written in Java, its integration into Dryad should be easy. However, many of the curation tools listed in the Curation Proposal are provided as a Java application and tend to lack an entry API to use them as a Java API library. There are at least two approaches to use a Java-Application-only curation tool within Dryad:

1. Wrap a command-line tool by java.lang.Runtime. getRuntime().exec(String command)

This textbook approach is a simpler way to accommodate a command-line tool into Java; demerits of this approach are (1) catching/utilizing logging message is not so straightforward; (2) lesser control over the behavior of a command line tool, i.e. a fixed set of command-line parameters are only ways to manipulate a given curation tool.

2. Customization (repackaging)

This approach modifies the original source code of a command line tool so that the tool as a Java library can be fully utilized by a Java class that calls it. While this approach can control most of the original functionalities of a curation tool and for some cases add new functionalities, it is relatively time-consuming and requires a skilled developer who can understand the original source code quickly and modify the entry point class according to given requirements.

When well-known curation tools mentioned in the Curation Proposal are integrated into Dryad, it is likely that these tools would use the first approach. A relatively new, omnibus curation tool, FITS (http://code.google.com/p/fits/), encompasses well-known format-identification and metadata-extraction tools, but its usage is limited a command-line tool. Another curation tool mentioned in the Curation Proposal, Xena (http://xena.sourceforge.net/), has format-conversion functionality, which FITS does not have, but it is available as a standalone GUI tool.