Software Release Procedures for 1.1

Jeffery A. Triggs & Dave Hoover

February 8, 2006

1 Introduction

This document supersedes the one by Michael Giarlo entitled "Software Specifications and Synchronization Procedures" dated November 1, 2004. It reflects our experience with the release to Systems of version 1.0 of the WMS, DLR, and NJDH projects during the late summer and fall of 2005. It is intended as a guideline for future releases, but specifically the release of version 1.1.

The following sections describe: 1) the standard programming practices to be used by developers to ease the migration of software from Linux development to Solaris production environments; 2) the base software platform, which should already be in place before any release of the project software; 3) the different projects to be released along with their functions; and 4) the release procedures specific to these projects.

2 Standard Programming Practices

It is assumed that the development platform for all projects is Linux and the production platform is Solaris. All modifications and bug fixes are to be done on the Linux development machine and then ported to the Solaris production machine using one of the methods described below.

The following recommendations, based on Michael Giarlo's list, were implemented for the 1.0 migration and remain in effect for future releases.

- Parameterize hard-coded references to hostnames, IP addresses, and filesystem paths.
- Combine like files into centralized folders.

- Relocate global and widely-used variables to a centralized configuration file or files.
- Establish error-checking routines wherever necessary.
- Remove all extraneous files once development is completed, such as backed up code, archived versions, old copies, unused code, etc.
- Separate “frozen” production code from development code. Development code and production code should never be kept in the same directory.

The following suggestions come from Dave Hoover and will be useful in streamlining the process in future releases.

- Use clearly defined test procedures on the development side to test usability and functionality of the release software.
- Make all programmers aware of log files where error and informational messages generated by their code reside. All error messages should be eliminated, or a document should be provided explaining why they occur and what they mean.
- In testing on the development side divide problems into those that need to be fixed before the release can go forward and those that need to be made as fixes in a future release. This should be noted in the MR system.
- All software changes on the development side should be tested long before the final release testing is done to minimize the amount of problems found.
- All programmers should be be aware that any new software, or compilation changes that are required by their code need to be communicated in a timely fashion to both the development and production UNIX admins so they can get the changes in place prior to the migration period.
- All programmers need to keep readme files up to date about their software listing any ownership and permission required for their code to work. It should also include any special installation steps that need to be followed.
- Once the production software is in place the testing team needs to step through the same defined test that was done on the development machine to assure that both sets of software are functioning the same.

- Problems unique to the production machine need to be looked at in terms of severity to see if they require a fix before we can move forward. Some production problems may lead us to change development code so that both systems can use the same code in the next release.
- All involved need to be aware of the decisions made by the SW_ARCH group and the Steering Committee so as to not rehash issues that have already been decided.
- All programmers and UNIX admins need to be available for the whole period that is defined for the testng/fixing/tranfer/testing period.
- All full time programmers should have some access to the production machine to assist in debugging problems that may arise in the migration of their code. This includes being able to access the error log files.

3 Basic Software Platform Requirements

This section deals with the underlying software platforms necessary for the 1.1 release. The software is developed on Linux at the SCC and released on Solaris at Library Systems. The basic software packages are compiled and loaded separately on each of these platforms, and should remain stable throughout the 1.x release cycle.

3.1 amberfish 1.0

- afadmin
- afindex
- afsearch
- fetch

3.2 CVS 1.11.18

3.3 DjVu DocumentExpress 4.1

3.4 DjVuLibre 3.5.14

3.5 Fedora 1.2.1

The current version of Fedora at the SCC is 1.2 and the version at Systems is 1.2.1. This discrepancy has not led to any known issues in our 1.0 release, but we feel that the 1.1 release is a good opportunity to bring the SCC installation into sync with that at Systems.[1] Fedora 1.2.1 has the following software requirements:

- J2SDK 1.4+
- MySQL 3.x or 4.x
- JAVA_HOME and FEDORA_HOME environment variables
- Fedora runs under a manually created “fedora” user and group.

3.6 Java J2SDK 1.4.1

3.7 ghostscript 6.52

- ps2pdf

3.8 lame 3.96

3.9 mp3wrap 0.5

3.10 MySQL 4.0.20

3.11 Perl 5.6.1

Perl should have the following modules installed:

- CGI
- CGI::Carp

[1] Our 2.0 release will involve the move to Fedora 2.1 among other major changes.

- Data::Dump
- HTTP::Request
- LWP::Simple
- LWP::UserAgent
- MIME::Base64
- SOAP::Lite
- Storable
- XML::Parser
- XML::XPath
- XML::XPath::XMLParser

3.12 PHP 4.3.10

The PHP build options include

- –enable-xslt
- –with-dom
- –with-dom-xslt
- –with-expat-dir
- –with-exslt
- –with-mysql
- –with-xslt
- –with-xslt=sablot

The PHP configure options include

- short_open_tag Off

- max_execution_time 3600
- max_input_time 3600
- post_max_filesize 1024M
- upload_max_filesize 1024M
- register_globals On
- SMTP localhost
- smtp_port 25
- sendmail_from nobodywww2.scc.rutgers.edu sendmail -t -i

3.13 sha1sum 2.0.21

3.14 shorten 3.6.0

3.15 UNIX Utilities

- awk (GNU Awk 3.1.0)
- cat (cat 2.0.21)
- chmod (chmod 4.1)
- cp (cp 4.1)
- diff (GNU diffutils 2.7.2)
- echo
- egrep (GNU grep 2.5.1)
- find (GNU find version 4.1)
- head (head 2.0.21)
- ls (ls - GNU fileutils-3.13)
- lynx (Lynx Version 2.8.4rel.1)

- mv (mv 4.1)
- pwd
- rm (rm 4.1)
- sed (GNU sed version 3.02)
- sort (sort 2.0.21)
- tail (tail 2.0.21)
- tar (GNU tar 1.13.25)
- uniq (uniq 2.0.21)
- wc (wc 2.0.21)
- xargs (GNU xargs version 4.1.7)

3.16 xsltproc

Using libxml 20609, libxslt 10109 and libexslt 807 xsltproc was compiled against libxml 20609, libxslt 10109 and libexslt 807 libxslt 10109 was compiled against libxml 20609 libexslt 807 was compiled against libxml 20609

4 Identification of Projects

4.1 dlr/EDIT/notification

This is the main Fedora interface. The dlr directory contains the public search functions for various portals, e.g., NJDH and EMM, with view only Fedora links. The EDIT directory is password protected and contains the management functions, including ingesting objects, indexing, searching, object editing, management of the collections database, handle creation, object validation, alerting services, and a new statistics reporting tool. It requires the following software platform elements: PHP, Perl, amberfish, xsltproc, MySQL, and various UNIX utilities. It uses the authentication database and auth.class.php located in the doc_root directory.

4.2 disseminators

This is a group of Fedora disseminators that perform functions such as creating tables of contents for books and other multi-divisional objects, or listing objects in collections. It uses PHP and Perl and makes use of the PHP XPath class located in doc_root/phpxpath.

4.3 emm

This is an e-learning module maker that allows users to collect objects from the digital library repository for use in learning modules and learning units. It has its own search interface and "shopping cart" mechanism. It uses PHP, MySQL, Javascript, XSLT, and Perl.

4.4 etd

ETD is a web submission solution for managing electronic theses and dissertations. It allows for thesis or dissertation submission by students, editing and approval by faculty, approval by the graduate school and final deposit into an institutional repository. The technologies used include PHP, MySQL, Perl, JavaScript, XML, and XSLT.

4.5 njdh

This is the public interface for the New Jersey Digital Highway Project. It consists of a set of static web pages maintained on the production server, but it also includes a version of the DLR search programs for searching and dynamic display of results. Thus it requires PHP, Perl, amberfish, xsltproc, djvudecode, and various UNIX utilities.

4.6 WMS

This is the main interface for creating the metadata needed by Fedora objects, creating presentation datastreams from archival files in a "pipeline", and creating the external XML objects used for ingestion into Fedora. It uses MySQL, PHP, Perl, the DjVu utilities, the ghostscript utilities, lame, sha1sum, tar, and other UNIX utilities.

5 Release Procedures for Projects

The original beta releases of these projects involved what Michael Giarlo called "snapshots" — a series of MySQL dumps and tar files of relevant directory trees. By the time of release 1.0, this method had at least partially given way to the use of a CVS repository into which developers committed code and from which the production site manager could download the code.

For future releases, we envision a more systematic use of CVS, which should ease the migration process considerably. CVS enables the maintenance of one or more stable codebases that will allow developers to work on different releases at the same time. It also allows complex versioning and recovery within the codebases on a file by file basis.

Not everything is most efficiently maintained in CVS, however, such as static web pages that will be edited on-site when necessary, or dynamically growing directories that are populated by user-generated data rather than code. The following sections will outline release procedures tailored to the needs of each project.

5.1 dlr/EDIT/notification

The dlr directory tree is completely stored in and delivered from the CVS repository. Individual files are updated as needed by the developer. The production site manager, however, will be downloaded the entire directory tree (what CVS calls a "module") into a staging directory for any new releases or updates. This staging directory can be used for a final test before the release is "staged" over the contents of the previous release on the production server.

5.2 disseminators

The disseminators directory is also stored in and delivered from the CVS repository. Individual files are updated as needed by the developer. The production site manager will handle these files the same as the dlr files.

5.3 emm

The emm directory tree contains user-created modules with uploaded files of considerable size. It is therefore not a good candidate for CVS, but will be maintained

on the production server. The application software for emm itself is relatively stable and will be delivered as a tar file if need be. The emm search hooks into the njdh search functionality, which will be maintained as described below in CVS.

5.4 etd

The ETD project is still in the early stages of development and has not yet reached a stable form. We expect that it will be delivered, at least in the foreseeable future, as a tar file.

5.5 njdh

The static files of the njdh site as well as the "site search" of such files are not stored in CVS, but as of the 1.0 release will be maintained on the production server. The "search" subdirectory, however, which hooks into the dlr search system, will be imported into CVS, maintained there, and delivered in a manner similar to that used for the dlr and disseminator directories.

5.6 WMS

The WMS directory tree is completely stored in and delivered from the CVS repository. Individual files are not updated, however. When a new release of WMS is ready, the entire tree is imported into CVS. The production site manager will handle these files the same as the dlr files.

5.7 Database Dumps, Resource Include Files, and cron jobs

In the 1.0 release, which involved the migration not only of code but of actual content from the development machine to the production machine, database dumps and include files of resources were of considerable importance. These files are not necessary, however, for future releases.[2] The development and production sites are expected to maintain their own databases and resource include files.

The development and productions sites use cron jobs to handle various tasks, such as digital signature testing and the nightly reindexing of XML objects in the

[2]Release 1.1 involves the addition of several new tables to dlrcollections. These are used by the alerting services and the new statistics service. They will be sent separately as SQL table creation schemas to be loaded into the working database at systems.

repository. These cron jobs involve the alerting services in the event of checksum failures. Like the database dumps and include files, the crons are a “one time deliverable” and should be maintained separately from the rest of the codebase by the system administrator for each server. Most of the cron jobs are shell scripts run nightly by the root user. The crons used by the release are:

In /etc/cron.daily:

- afindex_update - reindexes the Fedora objects for amberfish
- alerting_digest - sends alerting services digest messages
- fedora-signature-checker - checks signatures and sends alerting messages on failure
- tmpwatch - cleans up various tmp files at different intervals

In /etc/cron.fiveminutely:

- workareaperms - changes permissions on files in the workarea

The cron jobs cannot be tested using the normal test plan, but require special procedures that must be worked out between the test team leader and the system administrator.