Asynchronous Release - Alternative Option

First, a important note: This proposal is entirely a brainstorm from Tim, meant to generate response & discussion. It does not represent the views of DuraSpace. It is merely an idea. I appreciate comments (pro/con) as we start to brainstorm out the best way forward for asynchronous releases.

Definitions

In this document, this is what I'm referring to when I say "asynchronous" versus "synchronous":

- "Asynchronous Module" or "Asynchronously Released" - When a specific module has the ability to be released **separate** from the formal DSpace release process. For example, if we release DSpace "11.0" with some modules, and then a few weeks later release DSpace SWORD Module version "11.1" on its own, then that module is considered to have been released asynchronously. Please note that in my example, it could be possible that the DSpace SWORD Module was also released synchronously (as part of DSpace 11.0), prior to its separate asynchronous update.
- "Synchronous Module" or "Synchronously Released" - When a specific module is released as part of the formal DSpace release process.

Why an Alternative Asynchronous Release Process?

This proposal is my response to the following development proposals:

- Asynchronous Release - Proposes that dspace-api be split into modules, and it should be versioned separately/differently from DSpace.
- Restructure Trunk Projects - Proposes that we refactor **all** dspace modules out of Trunk, into /modules/ in SVN

Although I agree with the **idea** of asynchronous releases and refactoring some modules out of Trunk, I feel the above proposals could make some of our normal development processes more difficult, for the following reasons:

1. We've already seen discussions on lists (see, e.g. Services thread on dspace-devel) around confusion caused by some modules being only in the "/modules" area of SVN (e.g. Services)
2. I worry that the more separate module versioning we do, the more confusing Q&A on listservs may be. E.g., when answering a question on DSpace Services, how does one easily remember that Services 2.0.3 was in DSpace 1.7.1, but Services 2.0.2 was in DSpace 1.7.0. This may get more complex as more core modules are versioned separately
3. Although the **idea** of asynchronous releasing **everything** (i.e. all modules) sounds nice to Developers, I worry that it really won't get much interest from 95% or more of our users (who generally are not developers, though they may have part time of a developer on staff). In reality, maybe it'd be better to draw lines between what our users may wish to see released asynchronously versus as part of a "packaged" release.
4. If we move towards asynchronously releasing everything, I'm worried "quality control" may decrease. How would we, as a trusted group of developers, be able to ensure consistent quality control of all these asynchronously released modules (i.e. when/how does more thorough testing happen for asynchronous releases)?

Goals of this Proposal

The goals of this proposal are to re-think the idea of "asynchronous releases", specifically:

- To try to find a "line" we can draw between what we will release asynchronously (separate from normal DSpace releases), and what should continue to be released synchronously (i.e. only as part of a "formal release" process).
- To find a balance that will better fit the needs of our various types of users. How can we allow early adopters to get "quick releases" of some modules, while also packaging things up into a "formal release" for those who really are not as concerned about upgrading individual modules.

I'm specifically staying away from recommendations for how to reorganize SVN or how to refactor our APIs in this proposal, as I feel it's better to first figure out **what we need**, and then try to determine **what SVN organization and refactoring** can help us to achieve that need. (that being said, I do suggest a very general SVN reorganization below, but don't get too specific about it other than to say what may be in Trunk and what may be external to Trunk).

Background: Core Infrastructure Modules Vs. Web UI/Services Modules

When it comes to DSpace, UIs and web services interfaces change over time (and likely will continue to do so). But, what is more "persistent" and what requires tighter quality control is the "DSpace Core Infrastructure Modules/APIs" (This is just my own name for the primary APIs behind all our various web interfaces)

Currently, these "Core Infrastructure Modules" are a bit split up in our SVN. We have:

- **Core Infrastructure Modules**
  - dspace-api (the "main API") -> in Trunk SVN
  - dspace-services (Services API) -> in /modules/dspace-services in SVN
    (NOTE: Obviously, if we perform major refactoring of 'dspace-api', many other newly-refactored modules may end up in this list.)

I consider both of these APIs to be "core infrastructure", or "core APIs". You could also potentially include a few others (as there are a few "gray areas", like 'dspace-stats' or 'dspace-solr' (also both in /modules in SVN)).

Outside of this core infrastructure, we have various Web Services or UIs which humans or machines interact with in order to "get things done" in DSpace. These obviously include:

- **Web UI/Services Modules**
  - XMLUI-related modules
  - JSPUI-related modules

    - SWORD-related modules
    - OAI-related modules
    - LNI-related modules
    - SOLR Stats-related modules
    - Discovery-related modules
    - REST API modules
    - WebMVC-related modules (New UI work)

In my mind, this is worth **differentiating**, as I see these as two separate types of modules. I, personally, would question whether we truly want **all** types of Modules to be asynchronously released, or if some require much tighter quality control that a normal release cycle (synchronous release) offers.

3rd type of module = Extension modules?

There also may be a 3rd type of module, which is essentially "extensions" to any of the above modules. Some of these "extension modules" may include:

- any modules representing new DSpace Curation Tasks (these extend a Curation API, which is part of the "Core Infrastructure")
- any modules representing extensions to one or more UIs. For example, if someone created an extension module which provided extra themes to XMLUI or WebMVC.

Asynchronous Releases of Various Module Types

So, to get to my point here... A part of me wonders if what we may want to work towards is something similar to the following:

Main Proposal/Idea:

- **Core Infrastructure Modules** should all sit in SVN Trunk, and be released **synchronously**, as they constitute a new release of DSpace, and should likely be versioned together (to ensure they can be tested together, and always undergo proper quality control, etc.)
- **Web UI/Web Services Modules** - Most (maybe even all?) of these could potentially be stored external from SVN Trunk and released **asynchron ously**. Obviously, some of the primary UIs or Web Services would also be released synchronously alongside any newly packaged release of DSpace.
    - NOTE: Obviously, this would mean moving XMLUI, JSPUI, SWORD, etc. **out of Trunk**. This may or may not be controversial idea. But, to me it's more logical than storing "core APIs" outside of Trunk. Mainly because, we **know** that these various UIs/interfaces will receive proper attention/testing come release time (in the form of Testathons, etc). Plus, it could potentially open the door to allowing users to install DSpace and "pick & choose" which UIs actually get installed alongside it (rather than getting all UIs installed at all times).

An example may be the best way to understand this idea! What I'm talking about is the following sort of release process

Example, Completely Hypothetical Release Scenario
(*I've chosen to call this hypothetical release DSpace "11" so that it has no real meaning or implications. I could have just as easily called it DSpace "2011" or DSpace "Mountain Lion".*)

1. Suppose we were to release a DSpace "11.0" which includes a new features in our "Core Infrastructure APIs". It also comes with a pre-packaged 11.0 version of our "centrally supported UIs": XMLUI, REST, SWORD, Solr Stats, Discovery, and WebMVC (This is a hypothetical listing of "centrally supported UIs")
2. A third party team may release their own Ruby on Rails or PHP DSpace interface which is "compatible with DSpace 11.0". This code is managed /supported external to the Committers group (and may or may not exist in the central DSpace SVN).
3. Suppose 1 month later, we realize a significant issue in our 11.0 version of SWORD (maybe we didn't properly follow the latest specifications). Therefore, we immediately release a 11.1 version of SWORD asynchronously, and provide "easy install/update instructions" for those who want this immediate SWORD fix. (We could also call this bug-fix release of SWORD "11.0.1" – the main idea here is that we are able to release the SWORD module **on its own**, for those who want an immediate fix to just that module.)
4. Suppose a month or so later we realize we also have some XMLUI bugs/issues we'd like to get a fix out for. So, we release an 11.1 version of XMLUI asynchronously, for those who want these immediate fixes. (Again, this bug-fix release of XMLUI could have also been called 11.0.1 – the main idea is that now we have also released an updated XMLUI module separate from the "core" of DSpace)
5. Now, it's a few weeks later an we suddenly realize there's a small bug in one of our "Core Infrastructure APIs/Modules", which may affect all/most of our interfaces. Now, we decide to release an official DSpace "11.1" release – this includes updated fixes to the Core Infrastructure API (now versioned 11.1), along with pre-packaging of the same SWORD (see #3 above) and XMLUI (see #4 above) fixes we already released before. We also release updated 11.1 releases of each of our other "centrally supported UIs" (see #1 above). We send out an announcement encouraging everyone to upgrade to this official DSpace 11.1 release.
6. It's now 6 months or a year later. We decide we're ready for our next major release, DSpace "12.0". This new release adds new features to the Core Infrastructure APIs, so we release it as the next major version of the software platform (11.0 -> 12.0). These new features are also now enabled in updated 12.0 versions of our "centrally supported UIs": XMLUI, REST, SWORD, Solr Stats, Discovery, and WebMVC.
7. The same third party team who developed a Ruby on Rails or PHP interface for DSpace (see #2 above) now may release a new version which is "compatible with DSpace 12.0". Or, maybe they haven't gotten to that version yet, and they only have their interface updated to be "compatible with DSpace 11.1".
8. Finally, the process begins again...we can release 'bug-fix' versions of modules early as we locate bugs, and also package up a larger formalized 'bug-fix' release (if we deem necessary, or if Core Infrastructure APIs are affected).

The key point here is two-fold:

- When the CORE APIs/Modules need a change, that constitutes a new DSpace release (bumps up versions across all modules).
- But, if the CORE APIs do not require updates, all our various Web UI/Service Modules can release their own fixes asynchronously (and version them by updating a minor or sub-minor version number, or similar)

Comments / Thoughts / Additional Brainstorms welcome! I just wanted to capture this brainstorm somewhere, so that others can comment on it, and tell me whether it's a good or bad idea.