Technical Elements and Interoperability Worksheet

The long term goal is to make these into post-incubation questions. We have the view that for Sakai 3 the conversation will be something like: "This subsystem has been identified for movement from incubation to product development. As you work toward production consider the following questions. They are intended to provide a lens with which to focus on individual architectural and deployment decisions and help achieve the goal of useful, usable, maintainable code."

We are applying this worksheet to the projects currently in development of Sakai 2.7 in order to determine if the questions in fact meet that stated goal.

General Technical:

- **What kind of project is this?** Classical tool, My Workspace/synoptic tool, service, other?
- **What level of code review has been applied?**
  - Formal group code reviews? Pair programming in progress of code development? Automatic scanners/static code analysis? Are there any artifacts from the code review that could speed the technical review?
- **What architectural or technical documentation exists?** Please provide.
- **How many people have been working through the code?**
  - What is the lifetime of the code? How long has this code been around?
  - For example, Conditional Release has been around for 2-3 years. TinyUrlService has been around since March 2009. An extremely short life might indicate a rush to production without adequate community response. A long life might merit a close look at accretion of code and old-fashioned-ness of the user interface; longevity might indicate a need to 'freshen up.'
- **How long and where has it been in production?**
  - Extended time in production might be considered a proxy for formally completed quality assurance testing. Multiple years in production becomes a form of federated or de facto QA.
- **Is the project planning on independent releases?**
  - If this project plans to have an independent release cycle separate from the kernel, more attention must be paid to independent maintenance tasks. For example, the project would require two maintenance branches separate from kernel branches, one for bug-fixing and one for functionality modifications which have end-user training and documentation impacts.
  - Independent release sets a higher bar for the minimization of dependencies. The project demands on the kernel have to be kept at the pace of the kernel, rather than driving the kernel faster to meet the independent release cycle. If the project is expected to have high demands for kernel services, the first preference should be toward release with core Sakai instead of independent release.
- **What happens if an instance uses, then uninstalls the project?**
  - Is content created by the tool able to persist after the tool itself is removed? Should it be? Does the tool create a dependency upon itself from other tools such that the downstream tool would break if the current one is removed? What is the impact on search post-removal? Is the search cache updated to remove unviewable content?

Code Maintenance / Metrics:

- **How large is the project?** (Lines of code / modules)
- **How many non-kernel Sakai dependencies?** e.g. Assignment API, Gradebook API. Kernel dependencies are assumed and need not be itemized.
- **How many non-Sakai dependencies?** We (Product Council) will itemize the standard non-Sakai dependencies so people can answer this question quickly.
- **Are there core dependencies on libraries not typically used in Sakai?** (JWT, Wicket, Cayenne, etc.)
- **How tightly coupled is your tool with non-Sakai dependencies?** In particular are there persistence toolkits and UI toolkits which would require massive rewrite of the project code to swap out.
- **What is the persistence mechanism?** Check all that apply: SqlService, Hibernate straight up no chaser, Hibernate with magic, home-grown, extensively mixed-mode.
- **Any known JVM version dependencies preventing or forcing a Java 6 or Tomcat 6 upgrade?** Can we, the Product Council, identify some tools or checklists for identifying these issues? Is this, in the Product Council's opinion, too specific of a question for this criteria list?
- **Are there automated tests?** If not, are features documented well enough to test definitively?
- **Are changes logged?** Are feature requests & fixes logged as they happen? Is the change log included in the documentation?
  - If this is a fresh and shiny new project, is there a process established for feature requests, bug fixes and change logging?
- **Is the documentation up to date with the tool?** Bonus points for yes.
- **Are there kernel changes required to support the project?** In what kernel release will the changes appear?

Production Setup / Maintenance:

- **Are there any conversions, migrations or other administrative preparations required in order to start using the tool?**
- **Is there required configuration or does it work out of the box?**
- **Can complex or objectionable features be disabled?**
- **Does this tool depend on a file storage mechanism other than Content Hosting?** For example does it automatically create thumbnails, create cache files, or store uploaded content? If so how are these files handled?
- **Are there any special requirements for deploying this project in a cluster?** For example, does this project assume JGroups, shared network storage, UDP multicasting, or Terracotta server?
- **Was load testing performed?** What were the results? Is a load testing report available?
• Are there features that are database intensive, either in terms of large data or heavy CPU demand?

Interoperability:

• Does any other Sakai tool or service depend on this functionality? (Which?)
• Are there overlaps in functionality with other tools or services? (Which?)
• Is content created within sites? If so, are archive, import, search and duplicate features implemented?
• Are there any data feeds (RSS, XML, JSON) exposed via the Entity broker, sdata, or a one-off implementation?
• Does the work implement particular notable standards? For example IMS, OAuth, Java Community Process Specifications (JSRs) or other open web standards.