Sakai Developer Practices

INTRODUCTION

The Sakai code base has a life cycle that takes it through cycles of rapid development and periods of relative stability before a release. Sakai releases are derived from code located in the trunk of our code repository and any code changes committed to trunk must be performed with this in mind. For Sakai, major refactorings of code or code not destined for the next release is located in branches created outside the trunk.

Since trunk code will eventually constitute the next release, changes required in trunk should not have to wait for the creation and completion of branches; that said, in recognition of the importance of branch work, a change in trunk will neither block nor invalidate work performed in a branch that may be merged later. This is to ensure that the next release is not held up waiting for a branch to be merged and paves the way for the inclusion of simple fixes committed directly to trunk when warranted.

DEVELOPER ETIQUETTE

In Sakai self-governing teams of developers are responsible for different parts of our evolving modular architecture and tool set. Team members are granted commit access to different areas of Sakai's versioned code repository. Invitations to join a project team are made when potential new members demonstrate sufficient interest, commitment and proficiency to a project team's leadership.

Core Committers

Individual team members maintaining trunk code are known as core committers and are responsible to the Sakai Community for crafting software that benefits higher education. It is expected that committers write code that is capable of being understood, maintained, tested and extended by others. Committers are responsible for reviewing and, when appropriate, applying contributor patches in a timely manner for projects they oversee. They are also responsible for evaluating relevant JIRA issues (e.g., bugs or feature requests) and coordinating their response with other project members and the Sakai Project Coordinator. Committers must also ensure that project documentation is kept up to date and includes information about each team member for easy identification. Committers should monitor Sakai email lists relevant to the areas for which they are responsible and respond promptly to questions and issues raised on the list(s). Committers should also review relevant commit email messages and alert other team members if code commits appear incorrect.

Contributors

The Sakai Community encourages anyone with an interest in getting involved in Sakai development to consider becoming a contributor. A contributor is anyone who helps move the project forward. All committers should also be contributors but most contributors would not be committers.

Developer Etiquette

Sakai committers as well as contributors are expected to adhere to a set of general principles we term developer etiquette. Etiquette applies to all those who are part of the Sakai community, but the weight of etiquette lies heaviest on those developers with core commit rights. The operative goal here is to ensure that all members of the Sakai Community work together in a productive manner that results in a better code base and avoids activities that invalidates, belittles or destroys the work of others. A spirit of collegiality, mutual respect and sense of shared responsibility should govern developer relations at all times.

A basic set of principles guides our approach to development and developer interactions:

1. Every member of the Sakai Community will respect the efforts and contributions of every other member of the Community.
2. No commit will blatantly rewrite the code of another developer without prior agreement.
3. Code changes should aim to deliver the required purpose. For instance, wide-scale code reformatting of other people's code without agreement is not acceptable.
4. If a committer needs to work in an area of code for which they are not responsible, they will contact the lead of that area of code and/or the Sakai-Dev Discussion Group to inform and discuss their intended actions.
5. A committer that is responsible for an area of code, either as lead for that area, or as part of a team should communicate regularly with the team of committers working on that code.
6. Cross project changes require discussion on the Sakai-Dev list and may require the creation of a branch depending on the impact.
7. Care must be taken in every commit not to impede the work of others.

SAKAI CODE REPOSITORY

The Sakai Community maintains its source code in Subversion, an open-source revision control system. While Sakai grant's everyone read access to our repository, only recognized committers are granted write access to one or more Sakai projects.

EXPAND THIS SECTION TO INCLUDE INSTRUCTIONS ON HOW TO ACCESS, DO CHECKOUTS, CONTRIB SPACE, ETC.

TRUNK MANAGEMENT

OUTLINE OUR APPROACH IN GREATER DETAIL

BRANCH MANAGEMENT

Code branches are intended to permit committers and contributors to engage in experimental work or large-scale refactoring of existing code in isolation from trunk code intended for future releases or from work occurring in other branches.

Requests for branches can be made by anyone interested in participating in the Sakai Community. Branch requestors must be willing to maintain the branch and assist Sakai committers with bug fixing and other branch management tasks if required. Additionally, all developers with access to the branch must sign the Sakai Contributor License Agreement (CLA) to ensure that their branch contributions can be merged with Sakai trunk code if warranted. Typically, branches are created in the Sakai SVN repository but branches can also be located in external repositories. Irrespective of location, all branches are subject to the terms and conditions of the Sakai License. (See also: License Management Practice)

Branch requests

Branch requests start with a public announcement on the Sakai-Dev list that outlines briefly the rationale for the branch. The branch requestor should also alert our SVN administrators who handle the actual mechanics of branch creation at svn@collab.sakaiproject.org. Following the branch request an open comment period of one business day will follow before action can be taken on the branch request. Objections to the branch request can be lodged by any member of the Sakai-Dev Discussion Group. If objections cannot be resolved favorably within two business days as a result of discussion and consensus-building on the list, the impasse shall be resolved by a majority vote of eligible committers subject to the voting rules outlined below (see Voting).

If the branch request is approved an accompanying Jira SAK "branch" type issue must be created in order to track and document work on the branch. As work progresses branch managers are expected to record their progress and commits against the SAK branch issue created in Jira. Given that trunk code is continually changing, branch managers must also be prepared to merge changes in trunk into their branch in order to stay synchronized with the latest updates to the Sakai code base. Branch managers must also keep relevant committers and the Sakai Project Coordinator informed of progress in the branch.

Branch merges

Once work on the branch has been completed and tested, the code must be reviewed by Sakai committers with responsibility for the relevant sections of trunk to which the branch pertains. Following a successful code review the branch manager must alert the members of the Sakai-Dev Discussion Group via email to the list of an intention to merge the branch back into trunk. An open comment period of at least one business day duration must follow the announcement in order to allow Sakai-Dev subscribers time to pose questions or raise objections on the list to the proposed merge. Objections must be accompanied by a written statement detailing the reasons for the objection. If no objections are raised the merge is considered approved by consensus and the branch commit should be scheduled on a date and at a time convenient for the committer performing the merge.

If one or more objections are raised by Sakai-Dev subscribers and the objections cannot be resolved favorably within two business days as a result of discussion and consensus-building on the list, the impasse shall be resolved by a majority vote of eligible committers subject to the rules outlined below (see Voting). If the branch merge is approved, and all licensing issues are satisfied, a relevant committer will merge the branch back into the trunk and the branch will be deleted and the accompanying Jira SAK branch issue closed within two business days of the merge. If the proposed branch merge is denied committers with voting rights should work with the branch manager to address the objections raised by the vote.

After the comment period is complete, eligible committers shall cast their votes subject to the rules outlined below (see Voting). If the branch is accepted, and all licensing issues are satisfied, a relevant committer will merge the branch back into the trunk and the branch will be deleted and the accompanying Jira SAK branch issue closed within two business days of the merge. If the proposed branch merged is denied committers with voting rights should work with the branch manager to address the objections raised by the vote.

PATCH MANAGEMENT

The Sakai Community both appreciates and encourages the submission of code patches by developers who lack committer privileges to the areas they wish to address. Beyond addressing bugs or otherwise improving our code base, patch submissions represent an opportunity for the Sakai Community to increase the number of volunteer contributors by engaging directly with others in the process of improving our software. Our approach to patch management recognizes these dynamics.

Patch development

A patch is the output of the `svn diff` command on an edit set of that portion of the code base to which the issue the patch is addressing relates. Patches should not contain wide-scale reformatting of the code base but should clearly represent the changes that the patch is attempting to perform. Patches should be used to change limited areas of code that address small and well defined issues. If the change is large in nature, then a branch must be created to manage the work.

Developers interested in contributing a patch should first discuss their needs, ideas and/or solutions with members of the Sakai-Dev discussion list and with Sakai developers with commit privileges to the area of the code base to avoid duplicating effort should a patch already exists or the issue has been already been identified in our JIRA instance and is currently being addressed by members of the Community. If no patch or convenient workaround exists then create a Jira SAK issue at http://jira.sakaiproject.org, setting up a user account if attempting to log into our issue tracking system for the first time (see http://bugs.sakaiproject.org/jira/secure/Signup!default.jspa). Associate the SAK with one or more Sakai components, list the versions affected and include a description of the issue, including a brief outline of your proposed patch. List yourself as the reporter.

Patch acceptance

Patch acceptance is not automatic; all patches are subject to review and commit approval by the Sakai committer team with responsibility for the long-term maintenance and development of the code to which the patch applies. In addition, patch contributors must sign the Sakai Contributor License Agreement (CLA) before their patch can be applied. When developing your patch we highly recommend that you do so in consultation with one or more of the Sakai committers with responsibility for reviewing your submission. When you are ready for your patch to be reviewed attach a copy of it to your SAK entry and assign the issue to a relevant Sakai Committer. Consider setting a watch on the issue to track changes to the entry.

Committers assigned to review a patch are responsible for reviewing the submission in a timely fashion. If the patch is accepted it should be committed as quickly as possible; if rejected committers should work with the patch contributor to eliminate deficiencies or discuss alternatives. The patch review process should be conducted in an open manner and in a spirit of mutual respect that acknowledges a contributor's effort to address outstanding issues and a committer's long-term oversight and maintenance responsibilities.

VOTING

The voting procedures employed to resolve technical issues are inspired by the Apache voting model. Our approach underscores the value we place on heeding the opinions of those in the community directly responsible for the long-term maintenance and evolution of Sakai software. While all in the Community are free to express their opinions relative to proposed modifications to the Sakai code base, only those committers with responsibility for the future maintenance of the code affected by specific commits are accorded a binding vote.

As noted above, everyone who is a member of the Sakai-Dev Discussion Group is free to express their opinion on technical questions. In most instances, decision-making is arrived at by consensus and there is no need for formal voting mechanisms to gauge opinions. However, on those occasions when objections are raised during the open comment period that cannot be resolved within two business days by discussion and consensus-building; formal voting procedures will be inititated to resolve the impasse. In these situations, committers with responsibility for maintaining the impacted code will be polled and a decision rendered by majority vote.

The polling period will not exceed two business days after which eligible votes will be tallied by Sakai Foundation staff and the results published on the Sakai-Dev list and http://www.sakaiproject.org. Votes take the form of a simple numeric value and represent the following assertions:

- Affirm: 1
- Abstain: 0
- Deny: -1

Approval is by majority vote; however a minimum of three (3) affirmative (+1) votes is required to secure approval. All negative votes must be accompanied by an explanation outlining the reasons for denial. Ties will trigger a second comment period and a second round of voting of the same duration as the first. If second round voting fails to break the tie or muster the minimum affirmative votes, the proposed action shall be considered denied.