Smarter Balanced Assessment Consortium Recommendation

Smarter Balanced Assessment Consortium

Smarter Balanced Software Development Best Practices

A recommendation for the Smarter Balanced Assessment Consortium

22 August 2012

measured progress

Table of Contents

Summary

As part of the work under IT Systems Architecture (RFP-03), this document is intended to provide the development teams working on the various components of the Smarter Balanced architecture with guidance on the software development process. This document makes best practice recommendations for project management, requirements and acceptance management, code management, user experience design, quality assurance, and deployment management.

Objectives

With multiple development teams working on a project of this size and complexity, it is important to have some commonality in approaches and methodologies across the teams. It is expected that most, if not all of the teams will already be following many of these recommendations, and by having them in place will:

- Ensure a consistent development process across teams.
- Allow for coherent cross-contract collaboration.
- Identify and address potential issues earlier in the process.
- Make integration issues easier to resolve.
- Provide consistent progress tracking.
- Enable iterative delivery, incremental acceptance, and continuous integration.

It is recognized that every team is unique and will already be implementing processes that work well for that team. It should also be noted that the vendor for Test Delivery (RFP-11) would be responsible for deploying the system, so it will be necessary to include that vendor, the Architecture Core Team, and approval from the Architecture Review Board. It is expected that the development teams will self-monitor their adherence to these guidelines and the Architecture Core Team will work out any issues that arise.

1. Code Management

Code management describes the processes associated with the writing and maintenance of source code. For the purposes of this document, the definition is extended beyond the typical source control aspects to include a few development best practices.

Use a Code Repository

All code assets should be kept in a code repository. While the particular technology used for code management may not be important, the key requirement is atomic commits. Atomic commits ensure that a particular change in the source repository will map deterministically to a build. The ability to know exactly what changes went into a build is critical for purposes of tracking and reliability. A repository that has good support for branching and merging is preferred and the repository must be backed-up on a regular schedule.

Establish Code Guidelines

Having coding style guidelines in place will help improve the readability and maintainability of code. The guidelines should detail the naming conventions used as well as structure and style rules to be adhered to for all work.

Code for Quality

The quality of code is owned by the developers and assured by testers. The code should be constructed in a way that makes it easily tested via unit tests. Unit tests should be created and checked in to the repository along with its corresponding code.

Database Management

Changes to the database structure need to be added to source control in a way that allows for easy migration from version to version. The recommended approach to managing database schema versions is to leverage atomic commits in the source control utility and follow a model of immutable schema deltas. Immutable schema deltas allow determinate upgrades of the database by ensuring that there is a reliable way to upgrade the schema from one version to another without compromising database integrity. This is usually achieved by assigning a version to each database modification, a delta script, and guaranteeing that once a delta has been deployed, it will never be modified. This ensures a reliable database upgrade provided the deltas themselves do not alter database content, only schema.

Encapsulate Configuration Management in Source Control

It is useful to manage configuration variables, such as differences between deployed environments, in the source tree itself. Tracking configuration parameters in the source tree assures consistency between infrastructure requirements and source code changes.

Code Reviews

It is recommended that all code changes are peer reviewed to ensure quality, discover potential mistakes, and confirm that coding guidelines are being followed. Ideally the reviews are done before code is committed to the repository.

Cross Project Recommendations

The following is a list of code management suggestions specific to the Smarter Balanced work.

- Use *Git as* a common repository since the plan is to move assets to *GitHub*.
- Establish a common coding guideline to ensure consistency between projects.

2. Quality Assurance

With the demands for quality in this industry, and the growing complexity of software across all industries, a modern approach to Quality Assurance (QA) is required.

Automate Testing

Although there will always be a need for manual testing, the more test cases that are automated, the greater the ability to safely manage project quality. This takes discipline and a close collaboration between developers and quality assurance. It is recommended that QA become engaged early in the development process and even assist with unit test design.

Write Test Plans and Test Cases

Concise test plans should be created that clearly communicate the scope of the testing effort and should be created before testing begins. Test cases should align with test plans and be constructed in such a way that makes them easily repeatable. For manual cases, this means having clear instructions that can be executed by anyone on the team. Automated scripts should be designed to be as independent as possible and accessible to all members of the team. Test cases should be traceable to a bug or requirement.

Use an Issue Tracker

All issues, bugs, and change requests should be managed in a system that oversees workflow and is available to the entire team. All bugs should be given a unique identifier, titles that are easily identifiable, and contain clear steps on how to reproduce the observed behavior. Bugs should be given a priority based on their impact to the system.

Maintain a Controlled Environment

A separate environment should be established for the purpose of QA. This environment should be isolated from the development and production environments and should closely replicate the production environment.

Cross Project Recommendations

The following is a list of quality assurance suggestions specific to the Smarter Balanced work.

- Use a common issue tracker.
- Use the same automation tools.
- Create standards for test plans, cases, and reports.
- Establish a common prioritization scheme for issues.

3. Deployment Management

The following are recommendations on how to simplify deployment of components and make the process less prone to error.

Automate the Build

Each project should support a fully automated build process. Having manual steps increases the likelihood of errors during deployment. Every commit should trigger a build. It is recommended that unit tests be executed as part of the build process.

Build Binaries Once

Binaries should be compiled once and stored in a location accessible to the deployment mechanism. Deployments should use the same binaries for each environment.

Keep Builds Fast

Builds should execute fast so that issues can be detected quickly. Builds that run over night, and fail, interfere with valuable development time the next day.

Automate Deployments

Deployment of a build to any environment should be possible with a single command or a single click. The deployment process should be the same to every environment. Minimizing the manual steps associated with deployment makes the deployment process testable and repeatable, which in effect minimizes the chance of issues.

Verify on a Production Clone

In addition to deploying the same way to every environment, each environment should be as close to production as possible. Prior to the final production deployment, a deployment should be performed to an identical environment as production, especially when virtual machines are used for nonproduction environments and physical machines that are used for production. This will minimize environment related deployment issues. Smoke tests, or sanity tests, should be used to verify that deployments are successful and the application still functions as expected.

Cross Project Recommendations

The following is a list of deployment management suggestions specific to the Smarter Balanced work.

- Use common build and deployment frameworks.
- Create an environment to facilitate integration testing.

4. User Experience Design

User experience design (UXD) should be integrated into software development and other forms of application development to inform feature requirements and interaction plans based upon the user's goals.

Interdisciplinary Collaboration

Prior to creating any code on a project it is suggested that the UXD team collaborate with the stakeholders and the development team to identify the best approach to define the scope of work and understand any technical limitations of the project. This ensures that the team has common understandings from the beginning.

Prototyping for Usability Testing

In keeping with the Agile principle of reducing waste, prototyping is used to communicate design, acquire early user feedback, and reduce development effort. This allows validating, and if needed, revising the design concepts prior to engaging the development team.

Reusable HTML and CSS

Front-end developers should create a repository of reusable code so that all of Smarter Balanced projects retain the same look and feel for the user as well as enhancing efficiency for the developers when working on new aspects of the project.

Wireframes

Wireframes are a quick visual reference of the intended interface and interactions associated with a specific task. Use wireframes to help illustrate the expected end-user experience and to catch usability issues before actual development begins. It is recommended that a pattern library be created so that wireframe patterns can be easily accessed and shared.

Accessibility

All user interfaces should be designed with accessibility in mind and at a minimum should comply with Section 508 guidelines. http://www.section508.gov/

Cross Project Recommendations

The following is a user experience design suggestion specific to the Smarter Balanced work.

- Establish a style guide that can be used across projects.

5. Project Management

Project Management ensures that project milestones are met within budget and that unexpected surprises are minimized.

Maintain a Milestone Plan

Establish a plan of action that identifies a project's intent and tasks defined to reach the project's goals. A milestone plan helps to dissect a project into smaller, more manageable chunks and provides a way to measure the success and completion of a project. Milestones are a motivational tool that keeps the project and its team members moving forward. When setting milestones it is important that they are achievable and that appropriate action is taken if milestones are not met.

Identify Cross Project Dependencies

Cross project dependencies detail how a project is dependent on the completion of another project's deliverables in order to be successful. Continue to monitor and improve interdependencies throughout the project.

Iterative Delivery

Within software development, a project should be completed in repeated cycles (typically a 6-week lifecycle) or short segments of time. This will allow developers to take advantage of what was learned during development of earlier parts or versions of the systems.

Align the Workstreams

To achieve short delivery cycles it is important to establish cross-functional teams. Continue to find ways to bridge communication, foster collaboration where silos exist, and find ways to enable the process to progress smoothly.

Cross Project Recommendations

The following is a list of project management suggestions specific to the Smarter Balanced work.

- Align progress reporting.
- Establish common milestones across teams.

6. Requirements and Acceptance Management

Requirements management assures that the needs and expectations of a project are clearly captured, analyzed, and documented. Acceptance management is the process of ensuring that a system meets a set of mutually agreed upon requirements.

Central Repository

Requirements should be clearly documented in a tracking system that is visible to all team members. Each requirement should be reduced to a single entry in the system and be assigned a unique identifier and prioritized to communicate its level of importance.

Iterative Process

Requirements gathering and organizing should be an ongoing and iterative process. Each requirement should be reviewed and approved by project owners as it is created. Regular reviews of the open requirements to assess traceability and testability are recommended.

Definition of Done

The definition of done varies amongst organizations, but there should be a clear understanding as to when a project has been successfully completed and is ready to be delivered by all team members.

Review Panel

A review panel should be established to confirm that acceptance criteria have been met. Designating a delegate who will review the overall process is recommended.

Identify Impeding Issues

For a sprint or release, management teams should discuss all of the obstacles that stop them from delivering the project. Allow time in the schedule to address any issues that may arise during the acceptance process.

Cross Project Recommendations

The following is a list of acceptance management suggestions specific to the Smarter Balanced work.

- Use a single system for managing requirements.
- Establish a cross team review panel.
- Synchronize acceptance testing across vendors.