

Kuali Architecture and Development Standards

**December 2007
Version 1.2**

Technical Council Members

Aaron Godert, Cornell University
Brian McGough, Indiana University
Ralph Olstad, San Joaquin Delta
Wes Price, University of Hawai'i
Cathy Tan, Michigan State University
Anthony Potts, The rSmart Group
Arlene Allen, UC System
Mike Torregrossa, University of Arizona
James Smith, University of Arizona
Leo Fernig, University of British Columbia
Laran Evans, Cornell University
Andy Slusar, Cornell University
Jeff Bauer, Florida State University
Jim Thomas, Indiana University
Phil McKown, Indiana University
Phil Berres, University of Southern California
Sabari Nair, Coeus Consortium

Kuali Development Standards: Introduction

Our Charge and Who We Are

As the Kuali Technical Council (KTC), we are empowered by the Kuali Foundation to formulate the architecture that underlies an open source enterprise-level suite of administrative information systems for higher education. We bring a wealth of experiences and skills as designers and developers of administrative applications at our participating institutions. While we represent the IT and administrative systems organizations, we also attempt to account for the needs of higher education institutions in general.

How We Will Measure Success

Our collective experience confirms that the Kuali Architecture influences all of the project cycle: planning, analyzing, scoping, designing, developing, and testing the reference implementation. The success of this Architecture will be measured in how quickly and easily the Kuali community can accomplish each of these project phases and how effectively institutions can implement its customized version(s) of Kuali products.

Our Values

We are committed to the community source development model, and to the value of collaboration in producing a quality product that serves interested institutions well. To this end, we strive to identify and encourage the use of best practices and open standards. We believe that delivering a modular, pluggable, and scalable design will enable the success of the development team. We are dedicated to balancing quality architecture with meeting development deadlines.

Effective communication is *critical* – within our group and broadcasted outside of the group to the developers, the functional groups, the project managers, and the Kuali Board of Directors. We are convinced that the Kuali Architecture can and should be reflected in the project planning and management, the quality of the Kuali results, and the working philosophy of everyone involved.

Purpose of this Document

Eighty percent of the total lifetime cost of software goes to maintenance. The original author of the code often does not perform the maintenance. This is particularly true with open source projects. Applying standard development practices across the various Kuali projects will result in substantial maintenance savings for everyone. Good standards promote better quality applications, code reuse, and enhance portability.

The standards and tools should remain stable during the course of development on the project. New technologies will be introduced when there are proven efficiencies and benefits that outweigh the cost of training, migration, and time required. Proposed changes should be presented to the KTC for discussion.

What if a Particular Standard is Preventing Progress on Kuali Deliverables?

It is not the intent for the standards to prevent work from getting done. The intent is for the standards to provide more help than hindrance to the development process. If a situation should arise where a Kuali development team feels they must deviate from a standard, the first step is to communicate with the KTC. The council can then collectively arrive at a recommended approach. In some cases, there may be a way to solve the problem and still adhere to standards. In other situations, it may be decided that an exemption to a particular standard is justified. Either way, communication with the KTC and across development teams is *critical*. The KTC can take any newly adopted best practices and techniques that are added to the supported standards and ensure that those are then broadcasted across the various Kuali projects as they evolve.

For proposed revisions to the standards, see the *Governance Process* section of this document.

Kualii Development Standards: Methodology

In general terms, the Kualii development methodology is a lightweight, iterative approach to development that focuses on individual components that can be quickly developed and integrated into a larger application. Frequent communication and interaction with users is *required* in order for this methodology to succeed. By simplifying the development process and emphasizing frequent testing and feedback, the software product has a much greater likelihood of meeting the user's needs.

Another fundamental rule of the Kualii methodology is to KEEP IT SIMPLE. Developers should resist the temptation to add unnecessary complexity unless it is duly warranted.

The goal of this methodology is to create high quality code that meets the user's expectations and provides stable and easy to maintain systems for the future. Frequent interaction between developer and customer creates "shared ownership" of the end product.

Testing Processes:

Developers are highly encouraged to write consistent unit tests that help ensure system quality over time as developers make changes to fundamental pieces of code.

- Tests should be written for all services, and other code layers as needed to ensure software quality.
- The enforcement policy: Development Managers and the Quality Assurance manager will define appropriate test cases and ensure that their development teams meet the necessary level of testing for their module.

- When possible, test fixtures should be used for consistency and to ease long-term maintenance.
- Tests that change current data in the database must provide a recovery mechanism. This is common to tests that would change the data that would be used by subsequent tests. This standard currently is to use transactions to roll back the database work of the test.
- The goal of the unit testing is to provide timely feedback about the impact of changes to code on a continuous basis.
 - Ensure that bug fixes that have been made remain fixed
 - Ensure that desired functionality of components is not altered or broken based on changes
 - Provide examples of proper use of code within the systems
 - Ensure that changes by one team does not inadvertently effect another team.

Quality Assurance Testing

Functional testing coordinators are assigned to each module and are encouraged to explore and test the full set of functionality of their respective modules. In their testing coordinator capacity they will fulfill several roles.

- Define and coordinate the execution of testing scenarios to find any issues where the system does not meet specifications.
- Ensure quality of software through iterative rounds of testing, and issue reporting.
- Participate in releases through module sign off with project management. Having veto authority if module is not passing critical testing scenarios.

Kuali Development Standards: Architecture

The Architecture of Distributed Applications

The contemporary architecture of applications is a set of distributed, loosely-coupled components and services that provide distinct business functionality. A Service-Oriented Architecture (SOA) approach to Kuali development is recommended. This architecture supports service delivery via an enterprise portal framework and the presence of key infrastructure services like a single sign on solution, and an enterprise workflow engine.

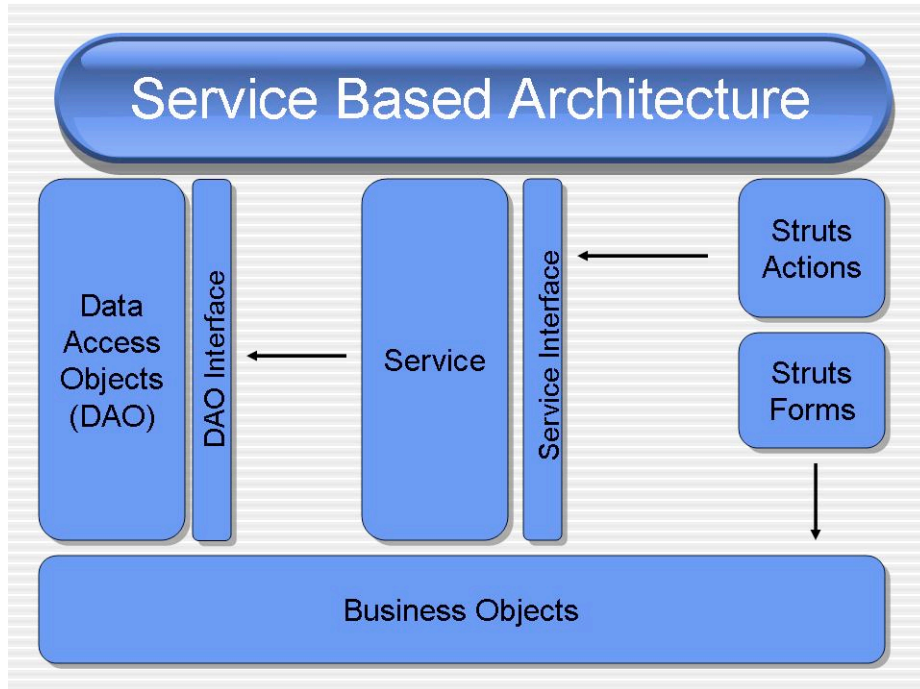
Components should be designed to build the application into three layers: Presentation, Business, and Persistence Layer. Components should be container agnostic and use standard J2EE web platform proven APIs such as Servlet and JSP. All system-to-system interoperability that cannot be done within the container should be done using web services technology or the emerging Kuali Service Bus.

The Presentation layer focuses on the presentation of the data to the user, e.g., screens rendered on a Web browser. The presentation layer should be constructed in such a way that future investment would enable alternate user interface implementations. To ensure

that alternate user interface implementations are possible, developers should avoid coding business logic into the presentation layer.

The Business layer provides the enterprise logic. It includes the components that capture the business rules of the enterprise.

The Persistence layer is the data repository for the application. It provides the data storage, retrieval, and maintenance. Data Access Objects (DAOs) should be employed to encapsulate the data sources using the chosen persistence mechanism to isolate access to the database. No business logic should be present in the data tier. No business object or presentation related object should access the database directly.



Kuali Development Standards: Environment Dependencies

- J2SE 5.0
- Servlet 2.4 Container
- JSP 2.0 Container
- Databases that support transactions through JDBC and integrate with the underlying persistence mechanism

Kuali Development Standards: Standard Libraries

The Standard Libraries provide the API to several frameworks and toolsets recommended in the Architecture section. JDK functionality should be used whenever possible rather than third party libraries.

- In particular, the currently supported standard set of Libraries and their corresponding licensing information are listed at: [Libraries and Licensing](#)

NOTE: The names of all dependent JARs that make it into SVN for the project should include the version number.

Kuali Development Standards: Migration/Upgrades

After a production release of a Kuali project, all future releases should offer an easy upgrade/migration path between versions. This should include scripts for update the database, migration paths for configuration settings, and documentation on changes that might effect schools that have customized any of the service layers.

Kuali Development Standards: Tools

While Kuali will strive to develop vendor neutral applications and services using a variety of tools and frameworks available to each team, it is recommended that a minimum set of standard tools and frameworks be identified for ease of code sharing and consistency in the code base.

The following tools and frameworks are used:

Java Integrated Development Environment (IDE)
[Eclipse w/Web Tools](#)

Source Code Version Management
[SVN](#)

Automated Continuous Integration
[Anthill Pro](#)
[Bamboo](#)

Application Server
[Tomcat](#)
[Sun JDK](#)

Web Server

[Apache 2](#)

Database Client

[Oracle](#) drivers

[MySql](#) drivers

Application Build and Deployment

[Ant](#)

[Maven 2](#)

Documentation Tools

[Confluence](#)

[Word](#)

[Excel](#)

Project Management/Bug and Enhancement Tracking

[JIRA](#)

[Project](#)

Load Testing Tools

[JMeter](#)

Code Coverage Tools

[Clover](#)

[Emma](#)

Data Modeling Tools

[Erwin](#)

[Visio](#)

Source Code Repository Viewer Tool:

[Fisheye](#)

Profiling Tool:

PerfomaSure

[YourKit](#)

Database

The standard for Kuali database development is to avoid usage of any database specific proprietary features. For this reason, we are trying to avoid the use of stored procedures, triggers, packages, and any database specific data types. The goal of this is to stay vendor neutral from a database implementation perspective.

Database tuning will only be addressed after development is done adhering to the standard toolset for Kuali.

Data Exchange and Messaging

The [Extensible Markup Language](#) (XML) should be used to maintain and communicate document content between documents and workflow. In addition XML is used to define the data dictionary for Kuali applications. And finally, XML is used for the configuration of many of the Kuali related libraries.

The Kuali Architecture supports and highly recommends the use of XML based Web Services as a means for interoperability between its various systems and other external systems.

Security Standards

Kuali applications must ensure that their user requests are both genuine (authentication) and appropriate (authorization). Only when user identity and user privilege are both verified should access be granted.

All secure Internet communications involving Kuali applications should be secured during transport over https (SSL).

All secure Internet communications requiring digital signatures will use the X.509 digital certificate public key infrastructure protocol.

We should ensure that posted form data is validated before processing or accepting into the database. We must assume that forms can be manipulated on client browsers and submit data that is potentially invalid, and we must protect against this possibility.

Where sensitive data is identified in the system, that data should be protected using the encryption algorithms that are available to Kuali applications. The goal of the encryption is to protect against loss of sensitive information in the case of a break in. The minimum out of the box encryption protocol is DES; however, the system easily allows for pluggable encryption protocols and the KTC highly recommends using the AES protocol in production.

Logging

Developers should incorporate logging into the design and coding of all Kuali applications. [Log4j](#) is recommended to provide multiple levels of logging: debug, info, warn, error, and fatal. Each logging level may be triggered and directed independently in a configuration file. Specific logging functions may therefore be changed without code changes or redeployment. Developers must make sure that they are not emitting sensitive data into the logs for the applications.

Version Control:

SVN should be used for version control.

Supported Browsers:

Our supported platforms for Kuali software are as follows:

IE – Latest major version (at least 6 months old) and service packs (Windows XP and Vista)

Firefox – Latest major version (at least 6 months old) and service packs (Windows and Mac)

Kuali Development Standards: Batch Environment

Any jobs developed by Kuali development teams will be Java executables that can be run in batch and can be executed in the application server as scheduled by OpenSymphony Quartz.

Kuali Development Standards: Style Guide

Refer to the [Kuali Style Guide](#) for consistency across UI elements.

Kuali Development Standards: Coding Conventions

AAP

Comment: This link is misleading, this is a documentation guide, cant find our style guide

References for Coding Conventions

“The Elements of Java Style” (Allan Vermeulen et al; Cambridge University Press, 2000; ISBN 0-521-77768-2) is recommended as a practical and comprehensive set of conventions for coding style.

Use of JavaScript

JavaScript should be avoided if at all possible. There are a few specific things where JavaScript is allowed: auto-focus, confirmation, XMLHttpRequest, and window.open. Any additions to this list are subject to review by the KTC. Kuali applications should still function properly with JavaScript turned off. JavaScript should only be used for “value added” features and should NEVER be required for an end user to complete a function.

Comments

Adequate documentation in source code files includes Javadoc comments. Comments should explain the intended function of the subsequent code and why it is present, in addition to how it accomplishes its function. Of course, any subtle or complex code should be explained using in-line comments. All classes and methods should contain Javadoc comments explaining the functionality contained in the class and the methods.

In addition, code that is not being used for any reason or that is commented out for a current release should always be commented on.

Kuali Development Standards: Process for Committing Code

It is up to the development manager from each contributing development team to ensure adherence to the correct form. Code reviews should be conducted frequently to ensure all developers are on the same page and that standards are being followed and that new developers can learn from the team.

Developers should ensure the following:

- Source code is tagged with appropriate licensing information
- All source code should compile and the web app should run before committing
- Library check-ins are managed as part of the project, including the source and Javadocs of the library being checked in
- Names of dependent JARs should include version numbers
- Adhere to naming standards
- Unit tests should continue to run at least at the same success rate that they were at before the change and before synchronizing with the latest code from HEAD just before checking in

The Kuali Foundation encourages and provides a certain level of support for contributed tools and applications that fall outside of the core Kuali code. Please see the [Kuali Code Contributions Guide](#).

Kuali Development Standards: Governance Process

Our goal is to maintain the same development standards over the course of development of Kuali systems. Changes to the standards and the addition of new tools will be discouraged for the sake of stability. Changes to standards will require compelling evidence that overwhelmingly displays the benefits to the Kuali projects in the form of more efficient development, better quality code, or other benefits that outweigh the cost associated with adopting the change. Exceptions to standards can only be made with the approval of the KTC.

Exceptions to Standards

In the event that a development team wants to make an exception to a standard, the first rule is that they must communicate their intentions to the KTC. If a simple majority agrees, via one vote per institution, the exception is approved. Urgent matters can be covered using the [KTC mailing list](#) if they can't wait until the next KTC meeting for a resolution.

Changes to Standards

In the event that a development team feels they have discovered a development approach or tool that can greatly benefit the overall goals of the Kuali projects, they may petition the KTC for a modification to the development standards.

At each KTC meeting, any petitions for modifications to the standards will be addressed. The petitioning group will present its case including arguments for how the change will benefit the overall goals of the various Kuali projects. Any group wishing to present a dissenting argument may do so. If a simple majority is reached and the majority of voting members are present, the change to standard is approved. Otherwise, the standards remain the same. All participants are obliged to accept these decisions as final for any work pertaining to the development of Kuali projects.