Application Development Code Review Procedure

Effective Date: July 28, 2015

1.0 INTRODUCTION

The purpose of this procedure is to guide successful peer review of internally-developed CSU, Chico applications. This procedure assumes a basic understanding of Github and using Source Controls systems. For more Information, https://www.google.com/search?q=github+tutorial&oq=github+tutorial.

The scope of this procedure includes all applications and/or code developed or managed by campus developers.


2.0 PROCEDURE OVERVIEW

2.1 Objectives

This procedure is designed to guide developers through an efficient and predictable code review utilizing Git, and CSU, Chico’s enterprise Github installation.

2.2 Benefits

- Increased service uptime through reliable code.
- Increased service quality through testing and review controls.
- Increased service security through review and scanning controls.
- Increased transparency/standardization of code across departments.

2.3 When to Use this Procedure

a. Before production deployment of all Application Code. In accordance with ICSUAM 8070.S000§1.6.1, applications must be reviewed before being placed into a production environment:

   i. “A code review of application code to locate potential security flaws and functionality problems should be performed before production deployment. Any security flaws found should be documented and tracked to resolution.”
3.0 PROCEDURE DETAILS

This procedure has a single distinct activity lifecycle but has several flexibilities at each step to account for department-specific needs.

3.1 Prerequisite Steps before a review: Design, Document, Initialize and Develop

Design

It is important to design application code around general programming standards so that the code is not only maintainable but also comprehensible for developer who may be collaborating or taking over the project in the future.

Many design patterns are dependent on the underlying technology choice. Developers should select the appropriate technology for the project with department standards taken into consideration. Further, Developers should consult campus developers and, if necessary, the Application Standards Committee, before utilizing new technologies. No new technology should be utilized without the proper authorization(s). However, do not disregard a design pattern solely on the basis that it has not been used by CSU, Chico previously. Each Developer should take into consideration reliable patterns as well as emerging patterns to ensure CSU, Chico remains technically relevant to industry standards.

Once a technology is chosen, apply proven software design patterns when applicable. If a third party framework is incorporated, make sure there is adequate proof of reasonable security. Examples: MVC, Repository Pattern, Factory Pattern

The development environments used for the project should be considered during Design time. Code environments and/or IDE should be properly licensed including all tools used to develop the application. Examples would be third-party libraries or frameworks.

Developers should also design data constructs in this stage as well. Determine where data is stored, the architecture of the project’s data, how it interacts with external data, and how it is accessed/transmitted. Service accounts should be established and Data accessed in accordance with the ICSUAM policy 8065 Data Access Standards.

Document

ICSUAM 8070.S500§1.1 requires that all Applications be properly documented prior to development begins.

Project charters that are shared by stakeholders can provide an excellent way to document projects. Consult the appropriate authority to obtain one of several general campus templates used by CSU, Chico. Creating an Entity Relationship Diagram (ERD) as you will need one for the technical specifications document that is required for all applications on campus.

Be prepared to respond to “Section 3 Documentation” of the attached Code Review Checklist during the code review.

Initialize

The application code to be reviewed must be in the CSU, Chico Enterprise Installation of Github in a repository as a distinct representation of the project.

E.g. Code must be in Github; One project per Repository.

The Code Reviewer(s) must have push access to the Repository containing the code to be reviewed. This is achievable by utilizing Github’s Team feature. Teams are local to each Organization. It is suggested that Team(s) be created for the sole purpose of Code Review in each technology such as “.Net Code Reviewers.” The team is then granted permissions to only the Repositories that they may be asked to
review. This strategy and practice better aligns to the ‘least-privilege’ ideals promoted by ICSUAM 8070.S000§1.5.6

**Develop**

As soon as a design has been selected, and documentation is complete, a developer may begin development. A developer should follow the design patterns previously chosen and documented. As changes are made that deviate from the original designs, the documentation should be updated to properly reflect the actuality of the project.

3.2 **Step 1: Initiating the Code Review**

Once the Developer considers the code a viable release candidate, the Developer initializes the Code review procedure by submitting a Git Pull Request. Documentation on how to submit a git pull request is available online by many websites detailing several different mechanisms. The CSU, Chico specific version is available at: [https://wiki.csuchico.edu/confluence/display/devbook/Home](https://wiki.csuchico.edu/confluence/display/devbook/Home)

3.3 **Step 2: Assign a Reviewer**

Assign a Reviewer to the Pull Request. This assignment step varies by Department.

3.4 **Step 3: Communication and Coordination**

The Developer communicates with Reviewer to either schedule a meeting to perform the review. This may or may not be face to face depending on available technologies, employee preference, and amount of code to be reviewed.

At this point, the Reviewer is provided with any additional documentation is needed to perform the review including background, purpose and additional commentary. This includes the When, Where, and How the review is going to be done.

**!!Note!!**

*Please remember to always use official university communication methods. Take necessary precautions of information security best practices. e.g. Do NOT give out sensitive data such as personal data, account names and/or passwords*

3.5 **Step 4: Code Review Time**

**What ‘NOT’ to do**

- **Design**
  - Code review is a process to review existing code and learn from it, *not* the time or place to discuss emerging design ideas. If design ideas come to mind, write them down separately and discuss them outside of the code review. Feel free to review the pre-determined/chosen design pattern, but do *not* get caught up in re-designing a project.

- **Write code**
  - If you are writing any code, you are not reviewing. The only exception would be a short line of two of code within a comment to help communicate a defect or suggested fix.

- **Judge**
  - Code review is a chance for everyone to learn and make better code while improving the code for the community. We all make mistakes and we all have opportunities to learn. Defects in code are an opportunity to learn and improve for everyone.
What to do
Using the Code Review Checklist, the Reviewer should analyze the code to complete the questionnaire. The Reviewer should document any questions/comments using the appropriate github comment section starting at code-level comment (on latest commit for that file) out to commit level comments and finally pull-request comments. This will pinpoint exactly where a developer needs to spend more development time. If everything looks good; say so! Documenting when things are correct (and then showing review) is helpful in verifying how accurately a review was done. This step should be iterative to allow feedback to the requestor and time to adjust and re-align the code to the new way of thinking. As small changes are made and committed, the same/original pull request is updated automatically by git.

3.4 Step 5: Finalize the Review
If, after reviewing the application, the code is determined to be a valid release candidate, then the github pull request should be accepted, merged, and closed by the Reviewer. Otherwise, the Reviewer should deny and close the git pull request. If denied, the Developer should incorporate the code review results back into the application code and re-submit a pull request once all improvements have been made.

4.0 COVERED SECTIONS

Several procedure activities outlined in this document seek to directly address requirements of ICSUAM Policy 8070.5000. Specifically:

- The entirety of this document defines a code review procedure to address section 1.4 (Application Coding) in which “Applications must be reviewed….before being placed into a production environment…”
- This code review procedure specifically utilizes the CSU, Chico Github installation to address section 1.4 (Application Coding) in which “The integrity and availability of source code … must be ensured by use of a source code control system…”
- The entirety of this document defines a code review procedure to address section 1.6.1 (Code Reviews) in which “A code review of application code…should be performed before production deployment…”
- The entirety of this document defines a code review procedure to be applied whenever any change is performed on application code to address section 1.7 (Web and Application Periodic Review) in which “Periodic risk assessment reviews should be performed … to ensure no new security risks have been introduced”

5.0 DOCUMENT SOURCES

This document is a compilation of original work as well as works researched online. The following articles provided significant contributions to the overall structure and content of this document:

6.0 CODE REVIEW CHECKLIST

6.1 General

- Does the code work?
  - Does it perform its intended function, the logic is correct etc.
- Is all the code easily understood (without comments)?
  - Comments are typically good, but code should strive to not fully rely on them.
- Does it conform to your agreed coding conventions?
  - These will usually cover location of braces, variable and function names, line length, indentations, formatting, and comments.
- Is there any redundant or duplicate code?
- Is the code as modular as possible?
- Does the code adhere to the chosen design pattern?
- Can any global variables be replaced?
- Are there any redundant or unused variables?
- Are all variables properly defined with meaningful, consistent, and clear names?
- Do all assigned variables have proper type consistency or casting?
- Is there any commented out code?
  - If code is commented out, it should be deleted. The magic of source control can bring it back if we decide we need it after all.
- Do loops have a set length and correct termination conditions?
  - Also check for speed; loops that can have many objects may be too slow.
- Can any of the code be replaced with library functions?
- Can any logging or debugging code be removed?
  - Debugging Code is suggested to be removed
  - Not to be confused with Application Usage logging which is require to be in place by CSUAM 8045.S600
- Does the code externalize configuration?
  - Look for things that are likely to change—number of items on a page in a paginated list, or the options in a dropdown list—and ensure they’re modifiable without changing the code. This also includes confirming that they are pulled from your standard location, whether that’s a database or a file on a file system, or wherever.
- Does the code perform Existence checks?
  - Before using objects, check that they exist. This will help put error handling close to the source of the problem. This is particularly true for anything obtained from a location outside the system (e.g., the response to a call to another system, or a file read off the file system).

- Is there any incomplete code?
  - If so, should it be removed or flagged with a suitable marker like ‘TODO’?
Some code will have todos; the job of the code review is to check that they are acceptable. For example, a “TODO: implement security restrictions” might be okay as long as it’s fixed before shipping, but it might not be okay if this release is going out the door tomorrow.

6.2  Security
- Are all data inputs checked (for the correct type, length, format, and range) and encoded?
  - Confirm that input from the end user is scrubbed and encoded.
- Where third-party utilities are used, are returning errors being handled?
- Are output values checked and encoded?
- Are invalid parameter values handled?
- Are there any known issues that have not yet been addressed that may show up when probed by a vulnerability scanner?

6.3  Documentation
- Do comments exist and describe the intent of the code?
- Are the “whys” documented?
  - Comments about why the developer did something unusual should be there. This is things like, “The first result from this third-party system is always null due to a bug on their end.” This also applies to complex functions or algorithms
- Are all functions commented appropriately?
- Is the code clearly and adequately documented with an easy-to-maintain commenting style?
- Are all comments consistent with the code?
- Is any unusual behavior or edge-case handling described?
- Is the use and function of third-party libraries documented?
- Are data structures and units of measurement explained?

6.4  Testing
- Is the code testable?
  - i.e. don’t add too many or hide dependencies, unable to initialize objects, test frameworks can use methods etc.
  - i.e. Some Testing Frameworks don’t allow testing of the constructor directly, so it’s best to keep the constructor slim and put any logic in a method that the constructor calls, which also renders that method testable.
- Do tests exist and are they comprehensive?
  - i.e. has at least your agreed on code coverage.
- Do unit tests actually test that the code is performing the intended functionality?
- Could any test code be replaced with the use of an existing API?

6.5  Error Handling
- Does the code comply with the accepted Exception Handling Conventions?
• Does the code make use of exception handling?
• Does the code simply catch exceptions and log them?
• Does the code swallow exceptions?
• Does the code correctly impose conditions for "expected" values?

Documentation Review & Approval

Review / Approval History

<table>
<thead>
<tr>
<th>Review Date</th>
<th>Reviewed By</th>
<th>Action (Reviewed, Recommended or Approved)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5/11/2015</td>
<td>Policies and Standards Group</td>
<td>Reviewed / Recommended</td>
</tr>
<tr>
<td>7/21/2015</td>
<td>Information Technology Executive Committee (ITEC)</td>
<td>Approved</td>
</tr>
</tbody>
</table>