GROUP TECHNOLOGY & OPERATIONS

Code and Build Guide (SDLC)

Global ID: DBS_11_G_0032_GR

Scope/Coverage: Group-wide

Issuer: Soo Lee Poh, SDLC Governance Team Lead

Associated Unit: Enterprise Architecture-SRE

Last Review Date: 01 October 2020

Review Frequency: Annually

Reference Policies, Standards, Guides:

No.	Global ID	Name	Document Type
1	DBS_11_S_0027_GR	DBS System Development Life Cycle Standard	Standard
2	DBS_11_G_0035_GR	Configuration Management Guide	Guide
3	DBS_11_G_0036_GR	Quality Management Guide	Guide

Associated Applications

No.	Application Code	Application Name
		Not Applicable

Associated Process Maps/Tools/Templates

No.	Name
1	SDLC Portal
2	Code Quality Guide
3	Code Release Plan
4	Unit Test Summary Report
5	System Operation Manual
6	Secure Coding Practice Guide
7	Secure Source Code Analysis Guide
8	Source Code Verification Checklist

CONTENTS

1 INTRODUCTION

1.1 Purpose

This guide provides a set of guidelines and best practices to support Development Teams in enhancing their application productivity and quality as they implement Code and Build activities.

It is recommended to use this guide as a supplement after reading the DBS SDLC Standard and Waterfall and Agile Procedures.

1.2 Audience

This guide is developed for: Developers, Development Leads, and Project Managers or Scrum Masters.

1.3 Scope

The guidelines and recommended practices included in this guide cover the following activities:

- Plan for coding activities
- Code development
- Code review
- Unit testing
- Code release

This guide does not cover all the industry's best practices. It is highly recommended that Development Teams practice whatever works best for their team and project.

1.4 Key Assumption

These guidelines are recommended practices only, and not how-to's or user guides. It is assumed that the Developers and Development Leads have the necessary experience and knowledge to effectively perform Code and Build activities.

2 CODE

2.1 Planning for Coding

Planning for the necessary coding activities is important to ensure there is a smooth progression from code development to review to its packaging and release. It also helps establish the following:

A clear understanding of the system requirements and design specifications: How you understand the requirements and design specifications directly impacts the accuracy and quality of the code you are going to develop.

A proper and efficient setup of the application development and continuous integration environment: This activity involves installation of code development IDE, OS, libraries, unit testing framework, etc. The continuous integration environment setup includes making use of the source control management tool, continuous build and test process with the available scanning tool. Make sure these activities are done according to the instructions defined for the application, including the correct tools and versions. Refer to DevConnect - Dev Tools for details.

Identification of coding guidelines to be used: This is to ensure there is a standardized way of doing code development within the team.

The term *Coding Guidelines* refer to the following:

- The Code Quality Guide provides high level attributes to determine quality coding and the code quality scan requirements;
- The Secure Coding Practice Guide provides insights into the hacker mindset, the potential goals and attack methods of an adversary and link to secure coding practices;
- The Secure Source Code Analysis Guide to recommend ways of finding security defects and fixing them before application is promoted to production; and
- Other application-specific or language-specific best practices that the Development Team has established internally.

Identification of the code review process: This will help ensure the code is secure and of quality. The code review may be done manually, or with the help of a tool.

A clear and established plan on how to deploy the code for testing: It is advisable to start preparing the Code Release Plan and System Operation Manual.

2.2 Developing Code

The code is the foundation of a system or application. It is always advisable to ensure that you apply and implement the defined coding guidelines to produce quality and defect-free code. Furthermore, make sure that you develop your code based on the following factors:

1. **Requirements and Design specifications** - Make sure the code complies with the specified requirements and design specifications that are approved for that system.
2. **Defined coding guidelines** - As each application is different, ensure your code adheres to the defined coding guidelines that are established for the application.
3. **Continuous integration and code reviews** - This ensures that the code being developed, and components used are of quality and free of security defects. Refer to Playbooks on Fortify, FOSS and SonarQube for details.

   For code developments, after the code scan, defects identified should be addressed as per respective quality gates.

For code developments that are not supported by any available code analyser system, the development team shall perform a full manual code review.

4. **Defined tools and techniques** - Using the properly set-up DEV environment, develop your code as per the defined tools, and in compliance with the techniques that are established for the application.

In addition, Development Teams can follow some of these best practices in developing or modifying their code:

Use of Coding Techniques	One example is the use of Pair Programming (commonly done in some Agile projects), which can be utilized for effective coding and review.
Writing Pseudocodes	Depending on the application need, a pseudocode can be written first for critical business functions.
Documenting Code	As a best practice, document the code with information, such as: • Description of the code • Date when it was developed • Name of Developer • Changes made, and the reasons for changing • Version number

2.3 Reviewing Code

Code review is a systematic inspection of a code. Considered as one of the most important activities in the Code and Build phase, it is designed to check and achieve code quality by covering the following aspects –

Code Reliability: Quality Code must be reliable to reduce and prevent application downtime, outages and errors that directly affect users.

Code Efficiency: Quality Code should be high performing and not use unnecessary resources. Any software response-time degradation may impact customer or user satisfaction.

Code Maintainability: Quality Code should be adaptable, portable and transferable.; and

Code Security: Quality Code must be securely coded. Poor coding practices, a bad design or wrong architecture might introduce software vulnerabilities.

Code Review Process

The importance of code review lies in being able to spot and fix defects early. The Development Team must perform a code review of their application when:

- There are new projects with new source codes; or
- The existing systems have changes in the source codes.

Code review is commonly achieved with the help of source code/software composition analyser systems, then followed by manual code review.

To perform manual code review, Development Team is required

- To pass the mandatory Code Review Assessment (Open Systems); and
- To use the Source Code Verification (SCV) Checklist to check for code quality and ensure code security.

Applicability of SCV Checklist for manual code review:

- Application where the Bank does own or have the source codes - Use the Source Code Verification (SCV) Checklist.
- Application where the Bank does not own and does not have the source codes - As the Bank does not own the source code, we cannot do the SCV Checklist validation. Instead, request a copy of the Product Vulnerability Report which consists of one or combination of SAST/DAST/FOSS/VAPT reports for the version of product used in the Bank. The report submitted shall be subjected to controls and Quality Gates as per the Bank's Standards. Refer to Product Vulnerability Report Playbook for details on scope, criteria and process.

NOTE:

- Development Team should perform an internal review to ensure good quality for script extensions (e.g., .bat / .exe / .csv / .sh) and configuration files (e.g., .xml / .xslt / .json / .yaml / .ini).

2.4 Unit Testing

Codes are tested at a unit level to ensure each piece of code performs the function it has been designed to do. Normally done on a single subroutine or module, it exercises common scenarios, such as normal operation, input and output validation, and error handling, to ensure they are working properly. While it may sound largely the same as code review, unit testing is not about checking every line of code or its coverage; instead it focuses more on checking the output of the code.

Unit testing is not done only to find functional defects, but to also verify that the code is doing what it is designed to do. Other factors to consider when doing a unit test are:

- **Objective** – know what is being tested
- **Dependencies** – each unit test should be self-sufficient
- **Naming** – name unit tests appropriately and accordingly
- **Test coverage** – perform positive and negative tests
- **Tools, logs, and results** – monitor unit test logs and results

3 BUILD

The build aspect of the Code and Build phase covers the packaging of the source code and its deployment to a testing environment. While this can be done automatically, it would help to consider a few reminders prior to triggering a code release. This is to ensure the package being deployed is compliant, complete, and includes the relevant work products, which will eventually help the Testing and Release Management Teams in performing their tasks.

3.1 Plan your code release(s).

It is always better to do this early, even before the actual development work starts—to establish the implementation process as well a clear timeline. This should include adhoc code releases, which are applicable to:

- **Configuration-type releases, which do not require any code changes (e.g., upload of an online document, refresh of graphic files, data patching, etc.)**
- **OS patches or fixes, without any application change**

3.2 Baseline and package your code.

Developers are expected to commit their code changes into their development branch regularly (preferably daily). When the code is ready to be tested on the DEV environment, merge all the developer branches into a single branch, and version (tag) them before triggering the build process in the DEV environment.

Prior to merging the code into the developer branch, make sure there is consistency in the configuration items to guarantee easy maintenance and reference.

File / folder naming guidelines	• Avoid extra-long folder names and complex hierarchical structures. Instead, use information-rich filenames. • Use the underscore (_) as an element delimiter. Do not use spaces or other characters such as: ! # $ % & ' @ ^ ` ~ + , . ; =) (• Elements should be ordered from general to specific detail of importance as much as possible. • Abbreviate the content of elements whenever possible.
Versioning guidelines	• Start the version control element with a **V,** followed by at least two digits, and placed as the last most element. • *Example*: If a major release is **V1.0**, then the minor enhancements are versioned as **V1.1**, and so on.

***NOTE**: Refer to the Configuration Management Guide for information.*

3.3 Use your tools.

To help with the source code packaging process, utilize tools that are designed to ensure you are checking-in / checking-out / compiling the correct and latest version of the source code.

3.4 Release the code.

Once the code has been tested on the DEV environment, it can be released for SIT, and then subsequently to UAT and Production. Use a release management platform to facilitate the release of your source code package.

Attach the necessary code release artifacts (i.e., SCV Checklist, release notes, code review scanning reports, sign-offs, etc.) when promoting the code to the next target environment.

3.5 Merge the branch with the trunk.

Merge the baselined branch, from where code is packaged and released into production, with the code repository trunk. This makes it easier for Developers to refer to latest code for the next code release cycle.

NOTE:

- Make sure the code selected to be ‘baselined into trunk’ is the same as that which has been versioned in the final build.

APPENDIX 1 ADDITIONAL REFERENCES

1. Fortify Playbook
2. Playbook on Open Source Software (OSS) Risk Management Tool
3. Product Vulnerability Report Playbook
4. SonarQube Playbook
5. Code Review Assessment
6. DevConnect - Dev Tools

APPENDIX 2 VERSION HISTORY

Version	Date of Issue	Summary of Key Changes
1.0	10 Mar 2017	Initial version
1.1	26 Jun 2017	• Removed Enterprise Tools.
1.2	29 Mar 2018	• Added reference to the Secure Coding Practice Guide.
1.3	29 Jun 2018	• Added OSS scan as part of development work, and reference to OSS Playbook.
1.4	30 Sep 2018	• Updated Section 2.1: Added new Coding Guidelines and removed language specific secured coding guidelines. • Updated Section 2.3: Revised the Reviewing Code emphasis. • Updated Section 2.3.1: Revised Code Review Process. • Removed Section 2.3.2: Removed Code Collaborative Activity Section (Moved to Code Quality Guide).
1.5	31 Mar 2019	• To include mandatory Code Review Assessment.
1.6	15 Apr 2019	• Change Associated Unit from ITSS to Enterprise Architecture-SRE.
1.7	2 Dec 2019	• Revised Code Review Process on applicability of SCV Checklist and Product Vulnerability Report.
1.8	01 May 2020	• Enterprise Code Quality Gate implementation.
1.9	01 Oct 2020	• Change OSS to Free and Open Source Software (FOSS). • Review frequency changed to annual basis to align with SDLC Standard