SAMS Project

Coding Standard Guidelines

Thomas Travers
Grantham University
CS406

SAMS Project

Thomas Travers

Coding Standard Guidelines

PURPOSE:

The purpose of this document is to set a standard and guideline so that expectations can be accurate as to the responsibilities and duty of each developer and team member working on the project. The standard set forth will also build consistency in coding style making it easier to review code and read through code as well as help reduce coding errors and reduce the bugs that slip through the cracks. Improving coding efficiency also increases our chances of reducing the time to market for product completion.

Required Items:

Required items will not be followed by an ""*

Optional Items:

Optional items will be followed by an ""*

File header *:

Begin files by describing attributes of its content (Tailor the amount of content in the file header to type of organization, and process used conventional, or agile)

File header format *:

File Location: Path to the file

File Name: Full path name of the file that contains the program including the device name / location.

Version number: The version number and release number (useful for versioning, maintenance, and software reuse)

Author name: Name of the programmer who creates the program file and writes the first version of the program.

Project name: Full name of the project for which the program is initially written.

Organization: Full name of the organization

Copyright: Copyright information

Related Requirements: A list of requirements implemented by the programmer
List of classes: A list of the names of the classes contained in the program file*

Related documents: A list of related documents, and their URLs if available.

Update history: A list of updates, each specifies the date of the update, who performs the update, what is updated, why the update, and possibly the impact, risks, and resolution measurements.

Reviewers: A list of reviewers and what each of them reviews*

Test cases: A list of test scripts along with their URLs if available*

Functional Description: An overall description of the functionality and behavior of the program-that is, what the program does and how it interacts with its client.

Error Messages: A list of error messages that the program may generate along with a brief description of each of them.

Assumptions: A list of conditions that must be satisfied, or may affect the operation of the program.

Constraints: A list of restrictions on the use of the program including restrictions on the input, environment, and various other variables (Kung, 2014 p. 452).

File contents:

List descriptions of sections, classes, operations in the file

Description of classes:

Class description – provide a functional description of each class
(a) Purpose – a statement of the purpose of the class
(b) Description of methods – a brief description of the purpose of each method, the parameters, return type, and other input and output such as files, database tables, and text fields accessed and updated by the method. The list should not include the ordinary get and set functions.
(c) Descriptions of fields – a brief description of each field including the name of the field, data type, range of values, initialization requirements, and values of significance.
(d) In-code comments-comments that are inserted at places of the code to facilitate understanding and tool processing (Kung, 2014 p. 451).

Naming conventions:

These protocols specify the rules for naming packages, modules, paths, files, classes attributes, functions, variables, constants, and the like. Naming conventions should help program understanding and maintenance. The names that are selected for the classes, attributes, functions, and variables should convey their functionality and semantics, and facilitate understanding of the program. Moreover, the names should be the same or easily relate to the names in the design. Meaningless names such as "flag", "temp", "x", "y", "z" and the like, must not be used. Also, names must not contain space, control characters, or special letters (Kung 452)

Naming convention formats:

Packages: Names must be meaningful and use lowercase alphabet only.

Modules: Names must be meaningful and use lowercase alphabet only.

Paths: All files should be kept in their respective directory paths. For example, source kept in the "src" directory. Subdirectories to organize files in a directory will be allowed if

(1) The directory has many files

(2) The sub directory must have a meaningful and generic names such as lib for all class libraries, or res for all resource files

Files: Names must be meaningful and use lowercase alphabet only.

Classes: Names must be meaningful and use camel case alphabet only.

Attributes: Names must be meaningful and use camel case alphabet only.

Functions: Names must be meaningful and use camel case alphabet only.

Variables: Names must be meaningful and use camel case alphanumeric only.

Constants: Names must be meaning and use camel case alphanumeric only.

Source code formatting rules:

Formatting conventions specify formatting rules used to arrange program statements. These include line break, indentation, alignment, and spacing. Most integrated development environments (IDE) define a default format, which can be changed per the needs of the software development organization. In Figure 18.3, the formatting conventions are illustrated (Kung, 2014 p. 453).

Source code formats:

Line break: To keep code neat and legible, line breaks will be used according to the standards set forth in the Linux Kernel variant of K&R style.

Indentation:To keep code neat and legible, indentation will be used according to the standards set forth in the Linux Kernel variant of K&R style.

Alignment:To keep code neat and legible, alignment will be used according to the standards set forth in the Linux Kernel variant of K&R style.

Spacing:To keep code neat and legible, spacing will be used according to the standards set forth in the Linux Kernel variant of K&R style.

Capitalization: Use of capital letters will conform to the standards set forth in the naming conventions.

Comments:

In-code comment conventions. If it is written correctly, in-code comments facilitate program understanding and maintenance. Thus, the coding standards should define the requirements and guidelines including the locations where in-code comments must, or should be provided, and the formats to write the comments so that tools can process them (Kung, 2014, p. 455).

Comments formatting:

Comments must be provided in the following places within the source code:
(a) Above a code line

comments must have the following format:
(a) For Java, C, and C++, preceded by "/" and ended with "*/"*
(b) For HTML, preceeded by "<!--" and ended with "-->"

comments must be used for the following purpose only:
(a) To describe or explain a block of code
(b) To describe or explain a function / method / class / object / template
(c) To display an information header
(d) To address or point out sections of code that are questionable and need attention
(e) To describe the entire file, it's contents, and it's purpose (at the beginning of the document)

Rules for implementing, changing coding standards:

Rules and responsibilities for creating and implementing the coding standards as well as reviewing and improving the practice must be defined. Before any change can occur, a developer must submit a request using a predefined request form. The request must first be reviewed and signed off by a team lead if not done by a team lead before it can be given to a project manager. Request will be evaluated and only approved if the change can make a meaningful and positive impact in compliance with the project requirements.