Coding and Git standards

CLEAN, READABLE AND CONSISTENT

Writing software is a team effort. However, without co-ordination and principles, a development project can quickly devolve into a chaotic mess from which complicated, inconsistent and unmaintainable software can result. As such, we have Coding Standards. These refer to a set of guidelines which specify how to write consistent, high-quality code.

Purposes of having coding standards

Coding standards ensure that code written by different programmers follows the same format. They ensure that the code produced is readable and can be maintained with ease. Coding standards allow for code reuse and (as stated before) ensure that the high-quality, understandable code is produced. Furthermore, the standards improve the efficacy of the project implementation from the stages of writing the code to debugging and detecting errors. Finally, having standards in place ensures that the cost of maintaining the code remains minimal.

Coding standards

Taken into context; we have been tasked with creating a simulation of a restaurant. This is a team effort which requires co-ordination and effective communication between members of the team. As such, to ensure we complete the project and produce work of excellent quality, it has been decided that the following standards should be adhered to:

1. ***Naming conventions for local variables, global variables, constants and functions***

    1. *Care should be taken to ensure that descriptive and understandable names should are assigned to the variables and functions used in the code.*
    2. *It is recommended that we avoid the use of numbers when naming variables. If numbers need to be used, then it would be appropriate to write the numbers as words and to keep the use of such to a minimal.*
    3. *Camel case should be used for variable names and functions, with a capital letter to distinguish between multiple words in a variable or function name.*

2. ***Ensure code readability***

    This cannot be stressed enough. The idea behind the standards is to simplify the work process of completing a project. As such, the following should be adhered to:

    1. *Write as few lines as possible. Do not overload a function with a copious amount of functionality. Rather, split the workload by making use of helper*

functions.

2. *Make use of indentation to ensure that your colleagues (or collaborators) are able to read and understand your code. The code should be written within indented braces.*

3. *Avoid writing repetitive code.*

3. ***Document your work***

Leave comments in the code explaining how the code works. What the code does can also be explained, however, it should be clear from the given specification what a particular function is meant to do. Use tools like Doxygen to aid in the documentation of your code.

4. ***Ensure that you have saved your work (make use of backups)***

1. This ensures that, in the unfortunate event that your workstation computer (or laptop crashes, you do not lose the work, time and effort that you have placed into writing your code.

5. ***Exception handling and return values***

When an error results either from the user side or undesirable behaviour, it is imperative that exceptions are thrown and handled appropriately. It is considered bad practice to ignore errors when they occur. Furthermore, for debugging purposes, make use of return statements that indicate whether the function being written produces the desired result(s). Typically, returning a value of –1, 0 or 1 should work well in most cases. Otherwise, make use of return values appropriate for the given situation.

Git standards

Merging and branching

Merging and branching are features of Git that enable developers to work on different, isolated, versions of the same codebase.

Branching – A Branch is a new/separate version of the main repository. To create a new Git branch, specify a name and (possibly) a starting point. This creates a new branch which stores a history of commits.

These branches are typically then merged back into a main branch upon completion of work. This means branching protects the mainline of code and changes in branches are independent.

Merging – Combining the changes made from multiple branches into a single, central or main branch.

A branching strategy is the strategy that software development teams adopt when writing, merging, and deploying code when using a version control system. (Haddad, 2022)

i.e., it is a set of rules that define how developers will go about interacting with a shared codebase, keeping repositories organized while multiple developers add various changes at the same time

Git Workflow

A Git workflow is a recommendation for how to use Git in a productive way, encouraging developers to leverage Git effectively and consistently. There are several such publicized Git workflows:

· Basic – Requires a single central repository. Each developer will clone this repository, work on the code locally, commit the changes, and push these changes to the central repository

· Feature branches – Makes use of branches such that each new feature in a codebase has a corresponding branch. In each feature branch, the new feature is built and tested; and when completed, the branch is merged with the master branch.

· Gitflow

This workflow makes use of two parallel, long-running branches

- Master: Main branch used only for releases. Contains only production ready code.
- Develop: Home of all completed and stable features ready. All new features are developed in separate branches and merged into develop when they are complete and tested.

From the develop branch, new feature (or release) branches are created. These branches are used to implement new features, handle bug-fixes, etc. Once these changes are complete, the branches are merged back into develop.

Commit message standards

A commit message on Git, is text added to a commit object (by the developer issuing the commit) which describes what changes the commit makes at a high-level. Additionally,

it includes the reason for these changes and any additional information.
General commit message guidelines include:

- Keeping the message less than 150 characters, using title case.
- Use imperative mood.
- Add a title that is at most 50 characters.
  - e.g. “Added Feature” and not “added feature”
- (Optional) Add a body – at most 100 characters.
  - Explain what the change is and why it was needed
  - Leave a blank line between the title and body
  - Separate body paragraphs using a blank line
  - Use hyphens (-) for bullet points

Code Reviews

Code (Peer) reviews is when a piece of code (submitted by one developer) is evaluated (by other developers) in order to ensure that the code being submitted is error free, does not contain inconsistencies, and follows best practices [such as those mentioned in the beginning of this report]. In addition to improving the quality of code, code reviews help in spreading knowledge amongst team members.

Typically, one developer evaluates another developers piece of code and, if the reviewer finds no issues, the code is then merged into the existing codebase.

In practice, code reviews are often done making use of GitHub – specifically using pull requests. The following is a standard, step-by-step process for a code review in GitHub:

1. *Create a feature branch* – The developer working on the code will create a feature branch in the GitHub repository.

2. *Write and commit the code* – The developer writes and commits code to this feature branch.

3. *Open a Pull Request (PR)* – The developer opens a **pull request**[1] on GitHub, selecting the source (feature) branch and the target (develop) branch. A title and description are added to the PR, summarizing changes made.

4. *Assign Reviewers* - The developer assigns one or more reviewers to evaluate the PR.

5. *Code Reviewing* – The assigned reviewers examine the code changes in the request by reviewing code changes in GitHub; leaving comments on specific lines of code which point out issues or suggest improvements; having general discussions about the code using comments. In this step, the developer issuing the PR might have to make changes – via additional commits – to the code, per the instructions of the request reviewer(s)

6. *Continuous Integration (CI) checks* – Automated tests may be run on the code changes, and the pull request cannot be merged into the development branch until these checks pass successfully.

7. *Merge the pull request* – Once the PR is approved, repository maintainers can then merge the request into the target branch.

8. *Delete the Feature branch* – This keeps the repository clean and is considered good practice.

Good practices in code reviews include the following:

- It is good practice for each developer working on an issue or code change to do so using a dedicated feature branch.
- It is advisable to name feature branches created as follows: “feature/<name-of-feature/fix>”
- Keep Pull Requests small. Such requests are easier to review and understand.
- Include a detailed title and description – the description of the Pull Request should explain the problem, solution, and important context.
- Ensure that the code in the request adheres to coding standards.

[1]A pull request, often abbreviated as PR, is a feature in version control systems like Git. It signifies a request to merge code from one branch into another, usually from the developer’s personal branch into a main or development branch (Gunnell, 2023).

- Include unit tests and documentation updates as part of your code changes.

References

Forsyth, M. (2022, March 26). Git Commit Messages: Best Practices & Guidelines. Retrieved from Initial Commit: https://initialcommit.com/blog/git-commit-messages-best-practices#:~:text=Follow%20these%20guidelines%20when%20writing%20good%20commit%20messages%3A,a%20body%20%28optional%29%20Less%20than%20100%20characters%20

Gunnell, M. (2023, August 10). *Pull Request*. Retrieved from Technopedia: https://www.techopedia.com/definition/pull-request

Haddad, R. (2022, March 8). git-branching-strategies. Retrieved from AB Tasty: https://www.abtasty.com/blog/git-branching-strategies/

Rose. (2022, March 8). *What is a code reveiw and why is it important for Code Quality*. Retrieved from LinearB: https://linearb.io/blog/what-is-code-review-why-its-important-for-code-quality#what-is-a-code-review

Schults, C. (2022, March 25). *The Best Way to Do a Code Review on GitHub*. Retrieved from LinearB: https://linearb.io/blog/code-review-on-github