# CSCI 1105 - Code Quality Conventions and Standards

## Documentation

Is the code well-annotated to ensure rapid understanding?

| | NAMES | HEADERS | COMMENTS |
|---|---|---|---|
| 1 LEVEL | Names appear unreadable, meaningless, or misleading | Headers are generally missing, redundant or obsolete; use mixed languages or are misspelled | Comments are generally missing, redundant or obsolete; use mixed languages or are misspelled |
| 2 LEVEL | Names accurately describe the intent of the code, but can be incomplete, lengthy, misspelled or inconsistent use of casing | Header comments are generally present; summarize the goal of parts of the program and how to use those; but may be somewhat inaccurate or incomplete | Comments explain code and potential problems, but may be wordy |
| 3 LEVEL | Names accurately describe the intent of the code, and are complete, distinctive, concise, correctly spelled and consistent use of casing | Header comments are generally present; accurately summarize the role of parts of the program and how to use those; but may still be wordy | Comments explain code and potential problems, are concise |
| 4 LEVEL | All names in the program use a consistent vocabulary | Header comments are generally present; contain only essential explanations, information and references | Comments are only present where strictly needed |

## Presentation

Is the code visually organized for a quick read?

| | LAYOUT | FORMATTING |
|---|---|---|
| 1 LEVEL | Old commented out code is present, or lines are generally too long to read | Formatting is missing or misleading |
| 2 LEVEL | Positioning of elements within source files is not optimized for readability | Indentation, line breaks, spacing and brackets highlight the intended structure but erratically |
| 3 LEVEL | Positioning of elements within source files is optimized for readability | Indentation, line breaks, spacing and brackets consistently highlight the intended structure |
| 4 LEVEL | Positioning of elements is consistent between files and in line with Java conventions | Formatting makes similar parts of code clearly identifiable |

## Algorithms

Is each part of the code as simple as possible?

| | FLOW | IDIOM | EXPRESSIONS |
|---|---|---|---|
| 1 LEVEL | There is deep nesting; Code performs more than one task per line; Unreachable code is present | Control structures are customized in a misleading way | Expressions are repeated or contain unnamed constants |
| 2 LEVEL | Flow is complex or contains too many exceptions or jumps; Parts of code are duplicate | Choice of control structures is inappropriate | Expressions are complex or long; data types are inappropriate |
| 3 LEVEL | Flow is simple and contains few exceptions or jumps; Duplication is very limited | Choice of control structures is appropriate; Reuse of library functionality may be limited | Expressions are simple; data types are appropriate |
| 4 LEVEL | In the case of exceptions or jumps, the most common path through the code is clearly visible | Reuse of library functionality and generic data structures where possible | Expressions are all essential for control flow |

## Structure

Is the code organized for quick understanding of parts and the whole ?

| | DECOMPOSITION | MODULARIZATION |
|---|---|---|
| 1 LEVEL | Most code is in one or a few big methods; variables are reused for different purposes | Classes are artificially separated |
| 2 LEVEL | Most methods are limited in length but mix tasks; Methods share many variables; Parts of code are repeated | Classes have vague subjects, contain many variables or contain many methods |
| 3 LEVEL | Methods perform a limited set of tasks divided into parts; Shared variables are limited; Code is unique | Classes have clearly defined subjects, contain few variables and a limited amount of methods |
| 4 LEVEL | Methods perform a very limited set of tasks and the number of parameters and shared variables is limited | Classes are defined such that communication between them is limited |

## REFERENCES


- Fitzgerald, S., Hanks, B., Lister, R., McCauley, R., Murphy, L. (2013). What are we thinking when we grade programs? In Proceeding of the 44th ACM technical symposium on Computer Science Education (pp. 471-476).
- Keuning, H., Heeren, B., & Jeuring, J. (2019, July). How teachers would help students to improve their code. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (pp. 119-125).
- Stegeman, M., Barendsen, E., & Smetsers, S. (2014, November). Towards an empirically validated model for assessment of code quality. In Proceedings of the 14th Koli Calling international conference on computing education research (pp. 99-108).
- Stegeman, M., Barendsen, E., & Smetsers, S. (2016, November). Designing a rubric for feedback on code quality in programming courses. In Proceedings of the 16th Koli Calling International Conference on Computing Education Research (pp. 160-164).