**Document 01 – Project Description & Requirements**

Contents

[1 Background 1](#_Toc135655395)

[2 System Requirements 2](#_Toc135655396)

[3 Domain Analysis 2](#_Toc135655397)

[4 Development Requirements 3](#_Toc135655398)

[4.1 Project Management 3](#_Toc135655399)

[4.2 Testing 3](#_Toc135655400)

[4.3 Coding 4](#_Toc135655401)

[4.4 Version Control 4](#_Toc135655402)

[4.5 Individual Team Members 4](#_Toc135655403)

[5 User Story Directions 4](#_Toc135655404)

[5.1 User Stories tab: 4](#_Toc135655405)

[5.2 System Tests Tabs 5](#_Toc135655406)

[6 Time Log Instructions 5](#_Toc135655407)

[6.1 *Individual Hours* Tab 5](#_Toc135655408)

[6.2 *Activity Codes* Tab 5](#_Toc135655409)

[6.3 *Time Log* Tab 6](#_Toc135655410)

[7 Sample Reports 7](#_Toc135655411)

[7.1 Sample Report 1 – User Story 12 7](#_Toc135655412)

[7.2 Sample Report 2 – User Story 14 7](#_Toc135655413)

[8 Getting Started 8](#_Toc135655414)

[9 Grading Rubric 9](#_Toc135655415)

# Background

A key management system (KMS) is needed to manage employees’ access to rooms in buildings where they work. Each employee has an electronic key-card which contains personal identifying information about them. When they encounter a door that is locked, they can insert their key-card into a card-reader on the door. The card-reader communicates with a central server to allow or disallow access. On the backend, the system provides services such as registering/removing employees, buildings, key-cards, assigning employees access to rooms (doors), *etc.* As well, various reports can be generated showing for example: who has accessed which rooms and when, *etc*.

Further information is found in the Domain Analysis section below. The User Stories for the system are in the *user\_stories\_KMS.xlsx* spreadsheet.

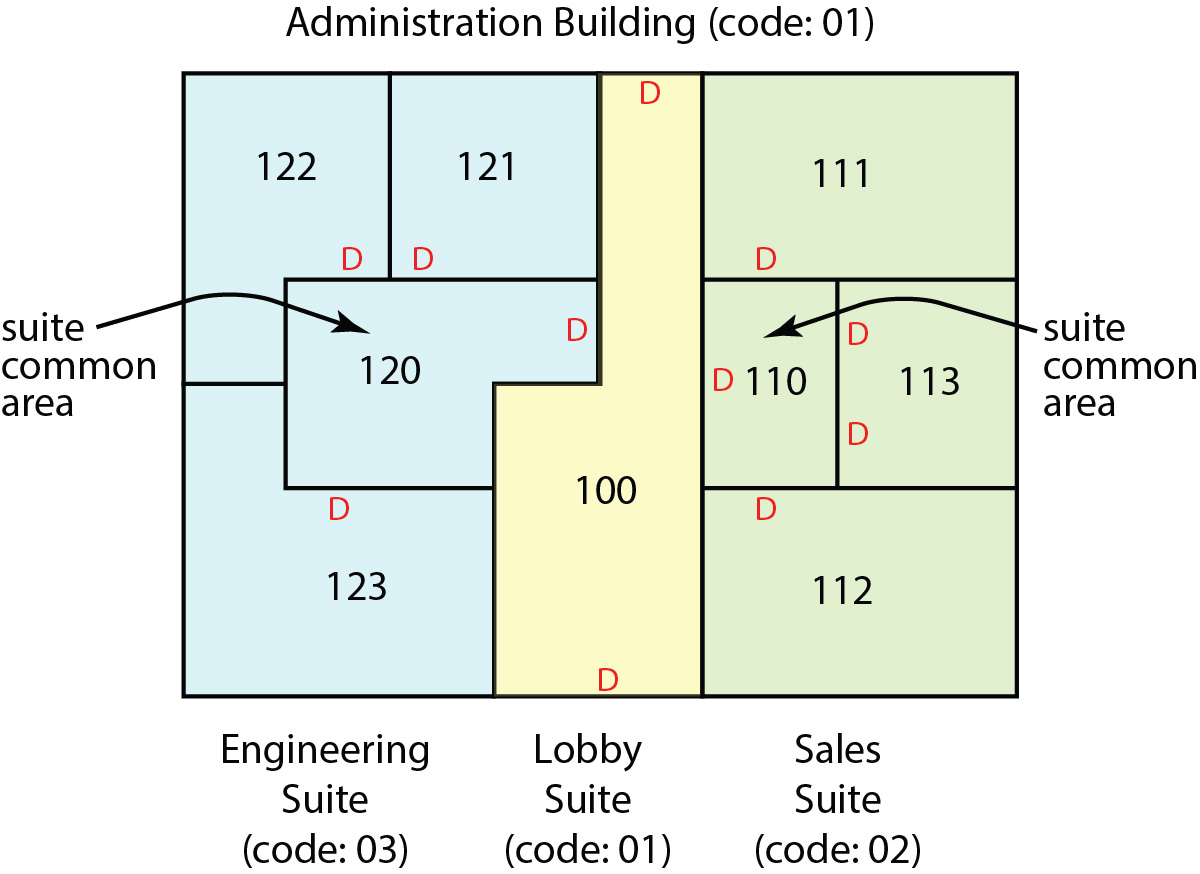
# System Requirements

1. The system is written as a stand-alone Gui application in Java. As described, this is an example of a client-server system. You will not write such a system. Instead, you will write a stand-alone, desktop system, a simulator of sorts, that enables all these functions via a single GUI.
2. The system is object-oriented (OO) and should utilize best practices for OO design.
3. The event handlers are as short as possible. In other words, they should call controller classes to accomplish the requested service.
4. The system utilizes text or binary files (or JSON) for data persistence. The system DOES NOT utilize a database.

# Domain Analysis

A building has a name (*e.g.* Headquarters, North Annex, Applied Manufacturing Center, *etc.*) and a unique 2-digit code (*e.g*. 01, 02, *etc.*). A building is composed of suites, each with a name (e.g. Accounting, Finance, Engineering, Atrium, *etc.*) and a unique 2-digit code (*e.g*. 01, 02, *etc.*). Each suite is composed of rooms where each room has a unique 3-digit room number (*e.g*. 172, 239, *etc.*), within a building, where the first digit represents the floor. As a simple example, consider the first floor of the Administration Building shown in the figure below and note the following:

* The first floor is comprised of three suites: Engineering, Lobby, and Sales. The “D”’s in the figure represent doors).
* The Engineering Suite has a common area that is designated as a room (120) which allows access to the doors to the main office rooms (121, 122, 123).
* The Lobby Suite is composed of just one room (100) which is the main entrance to the building, providing access to the Engineering Suite, through room 120; and to the Sales suite, through room 110. Additional floors would be similar, as well as additional buildings.
* The Sales Suite is similar to the Engineering Suite as it has a common area (Room 110) which provides access to the other rooms (111, 112, 113) in the suite.



Each room may have multiple doors, but doors are keyed the same so that an employee, with access to a room is able to use any of the doors. There are three levels of access that an employee can have.

|  |  |
| --- | --- |
| Access Level | Description |
| Room | Access to a single room(s) |
| Suite | Access to all rooms in a suite |
| Building | Access to all suites and rooms in a building |

There are two types of employees: Employees and Managers. An employee or manager has a first and last name, a unique 4-digit employee ID, and a password. An employee can have access to any number of rooms (or none at all). A manager can have access to any number of rooms, suites, and/or buildings (or none at all). Thus, an employee can multiple levels of access. For example:

* Sandra is an Employee and has *Room* access to rooms 111 (her office), 110 (the common area for the Sales suite), and 100 (the building lobby). Thus, she can come in the front (or back) door, enter her suite and then her office (111).
* Walter is a Manager and has suite access to the Sales Suite and room access to room 100 (the lobby). Thus, Alex can enter any room in the Sales Suite.
* Yolanda is a Manager and has building access to the Administration Building, meaning that she can have access to any room in the building. In addition, she has room access to room 833 in the Manufacturing Building (not shown in figure).
* Jeremiah is an Employee and has access to room 110 only. Thus, he can only enter the main building lobby when it is unlocked or a security guard is present (*i.e.* during business hours).

Note: these examples are used in the [Sample Reports](#_Sample_Reports) section below to specify the format for various reports.

# Development Requirements

## Project Management

1. You will use a GitHub *Project* to manage your development. You will have these columns (or more/others if you choose): *To Do, In Progress, Complete.*
2. For your project you are required to break work down into small tasks that 1-2 people will work on (some tasks may involve the whole group). These tasks should be estimated to take between 0.5 hours and 4 hours in length. **ALL tasks must be entered in your GitHub Project. Do not do work on the project without a task.**

Think about this carefully keeping in mind that you must work as a team, and that all team members should contribute to the development tasks. Tasks will be entered as *Issues* in GitHub and added to the *To Do* column. From there, they will be moved to an *In Progress* column and finally to the *Complete* column.

This will be new activity and it might be a bit of a struggle to do this. Almost all software companies use this approach. Many times, a project manager will develop these, sometimes in conjunction with developers. In some companies, a developer will simply go to the *To Do* list and “pull a ticket.” When complete, they pull another one.

Task/Issue breakdown is iterative. Your group will identify some initial tasks to get started on development. As issues are in progress or completed you will learn more about what you need to do. If a task turns out to be much bigger, or involve things you didn’t initially think of, then simply add them as new tasks, possibly closing out the initial task. Thus, Task/Issue breakdown is ongoing.

Sometimes a task may be to figure out what needs to be done, in other words, a planning session. The conclusion of that task will generally result in concrete tasks that contribute to the development.

Sometimes a task may be a placeholder of sorts, broad, open-ended, just so you don’t forget out it. Then, when ready to address it, it might be broken down into concrete tasks. In this case, I think I would remove the initial task once it has been broken down.

## Testing

1. You are required to have JUnit tests for each class following the conventions discussed in class.
2. You are required to define *system tests* that test each of the user stories. **Each user story should be tested in as many sets of circumstances as needed to verify that your implementation is robust.** These can be manual tests. Each User Story system test(s) is documented on a separate tab in the *user\_stories\_KMS.xlsx* spreadsheet.

## Coding

1. Programmed in Java using best practices for object-oriented software development: meaningful class and member names (and proper format-camel case), consistent indentation, naming of collections, naming booleans, encapsulation, class design, no static methods or variables (if you think you need them, submit your reasons to me immediately for my review), etc.
2. Adhere to the design principles presented in class (week 3, near the end: OO Design video and handout). The example presented in class only utilized one controller. However, keep the Single Responsibility Principle in mind with respect to controllers. In other words, you can have as many controllers as you need.

## Version Control

1. Only working, tested code should be in your master branch of GitHub and it should be in a package(s).
2. Each person is required to work in their own branch. Do NOT delete branches after merging/rebasing. Branches are required to be named: *lastname\_1, lastName\_2,* etc.
3. Commit early and commit often. When you add a method, commit. When you change something, commit. When you rename some files, commit.
4. The title of all commit messages is required to be prefixed with “ADD”, “FIX”, or “CHANGE”. For example: “Add *short title”*, “FIX *short title”*, “CHANGE  *short title”*. If you find you need another prefix, then use it. Another two may be: REFACTOR *short title*, MERGE *short title*. Titles should meaningfully summarize what was committed.

A commit message should detail why something was done. A commit message is not always necessary.

1. When you have code on a branch ready to merge with master, you should use the rebase workflow and then open a pull request. At least “some” pull requests are required to be reviewed by at least one other member before merging. In practice, all pull requests are reviewed. The more the better for your grade.

## Individual Team Members

1. All time spent on the project is entered, on a timely basis, into a spreadsheet I will share with your group. Directions for the Time Log are in [Section 6](#_Time_Log_Instructions).
2. You should work consistently on the project. In other words, you should not do a minimal amount of work one week, and then double the next week. This ensures that project is not held back.

# User Story Directions

This is information about what you need to do to the *user\_stories\_kms.xlsx* spreadsheet as your work progresses.

* Store the *user\_stories\_kms.xlsx* spreadsheet in your GitHub repository in a folder named *docs.*

## User Stories tab:

1. Do not add or remove, or renumber any user stories.
2. At the end of the sprint it should be sorted on Priority.
3. For the *Code* column (column H), type “C” if the code is complete, “~” if the code is in progress. Leave blank if you have not started the code.
4. For the *System Tests* column (column I), type “C” if the system test(s) have been written (in the spreadsheet). Otherwise, leave blank. The System Tests will be written in separate tabs, details are in Section 5 below.
5. For the *Status* column (column J), type “C” if the system tests have been run and are all passing, type “N” if at least one of the system tests is not passing, leave blank if the system tests have not been conducted.
6. For the *Comments* column (column K), write any comments that are needed about the status of the code or tests.

## System Tests Tabs

You will write system tests for each User Story and each test will be on a separate tab. There are two such tabs in the document already, just as samples. Thus, create a tab for each user story. The tabs should be named: US-1, US-2, *etc.* where the number represents the priority of the user story. On each tab, specify the system test(s) in the format below. An example is provided on the US-1 tab (which should be removed before turning in). There should be at least on system test for each user story.

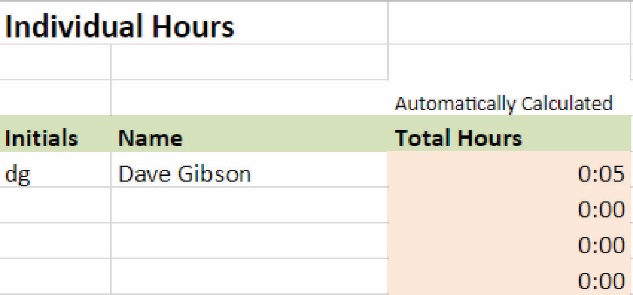
|  |  |
| --- | --- |
| **US Title** | [User story title] |
|  |  |
| **Test Num** | 1 |
| **Description** | [Brief description] |
| **Directions** | [List steps to conduct test so that it is reproducible] |
| **Expected Output** |  |
| **Comments** |  |
|  |  |
| **Test Num** | 2 |
| **Description** |  |
| **Directions** |  |
| **Expected Output** |  |
| **Comments** |  |

# Time Log Instructions

You will record all time that you spend for this project (including thinking, meetings, *etc*) using the Time Log spreadsheet that will be shared with you on Google Drive. Your team will share the spreadsheet. The spreadsheet has three tabs.

## *Individual Hours* Tab

The **second** tab is “Individual Hours” as shown below. The first time you use the spreadsheet, you should type in your initials and name (the first person should replace my entry). Your total hours will be automatically calculated. Do not alter that field.



## *Activity Codes* Tab

The **third** tab is “Activity Codes” as shown below. You will use one of these codes to allocate each increment of time that you contribute to the project.

|  |  |
| --- | --- |
| **Activity Codes** | |
|  | You will record all time (including thinking, meetings, etc) that you spend for this project using the Time Log tab on this spreadsheet. Use the Activity Codes below to indicate how your time was spent. |
|  |  |
| **Code** | **Task** |
| D | Any activity related to the analysis or design |
| C | Any activity related to coding |
| T | Any activity related to testing |
| P | Any activity related to presentations |
| O | Any other activity |

## *Time Log* Tab

The **first** tab is “Time Log” as shown below. Each time you work on the project you will make an entry by typing your initials, begin date and time, end date and time, interrupt time, activity code, and brief description of what you did. The total time and running total time will be automatically calculated. Do not alter those fields.

The date/time field is formatted; however, you don’t have to enter these exactly as shown. For example, for the first entry shown below, typing: “6/5 9 am” will produce the result shown.

Interrupt time is time you were not working. For example, you may have worked from 11 am to 1 pm, and taken 20 minutes for a snack. Thus, for interrupt time you would enter: “0:20”. For this to work properly, you must enter “h:mm”.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Time Log** |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  | Automatically Calculated |  |  |  |
| **Initials** | **Beg Date/Time** | **End Date/Time** | **Int.Time** | **Tot Time** | **Run Total** | **Act Code** | **Description of Activity** |
|  | 06/05 09:00 AM | 06/05 09:00 AM | 0:00 | 0:00 | 0:00 | D |  |
| dg | 06/05 09:00 AM | 06/05 09:06 AM | 0:01 | 0:05 | 0:05 | D | Project overview in class |
|  |  |  |  |  |  |  |  |

# Sample Reports

The following sample reports reference specific Use Cases in the Use Case spreadsheet.

## Sample Report 1 – User Story 12

List of all employees showing their name, ID, and rooms they can access and the corresponding, Building name and code, and suite name and code. For example, if someone has suite access, then all the rooms in the suite should be listed. In other words, a list of all rooms an employee can access. For example:

Sandra Johnson, ID=1234:

Room 100, Administration Building (01), Lobby Suite (01)

Room 110, Administration Building (01), Sales Suite (02)

Room 111, Administration Building (01), Sales Suite (02)

Walter Perchoy, ID=3324:

Room 110, Administration Building (01), Sales Suite (02)

Room 111, Administration Building (01), Sales Suite (02)

Room 112, Administration Building (01), Sales Suite (02)

Room 113, Administration Building (01), Sales Suite (02)

Room 100, Administration Building (01), Lobby Suite (01)

Yolanda Jeera, ID=4321:

Room 100, Administration Building (01), Lobby Suite (01)

Room 110, Administration Building (01), Sales Suite (02)

Room 111, Administration Building (01), Sales Suite (02)

...

Room 120, Administration Building (01), Engineering Suite (03)

Room 121, Administration Building (01), Engineering Suite (03)

...

Room 833, Manufacturing Building (08), Pre-fab Suite (11)

## Sample Report 2 – User Story 14

List of all employees showing their name, ID, and access levels. This is similar to the report above, but doesn’t break out all the rooms for Suite and Building access

Sandra Johnson, ID=1234:

Room access: Room 100, Administration Building (01), Lobby Suite (01)

Room access: Room 110, Administration Building (01), Sales Suite (02)

Room access: Room 111, Administration Building (01), Sales Suite (02)

Walter Perchoy, ID=3324:

Suite access: Administration Building (01), Sales Suite (02)

Room access: Room 100, Administration Building (01), Lobby Suite (01)

Yolanda Jeera, ID=4321: Administration Building –

Building access: Administration Building (01)

Room access: Room 833, Manufacturing Building (08), Pre-fab Suite (11)

# Getting Started

This is what I recommend you do to get started on the project.

1. Individually, or as a group do the following (I’m assuming you have already watched the video for these documents, now you should go deeper):
2. Read this (*Description & Requirements*) document carefully.
3. Read the *Sprint Report* document carefully.

As you do these, take notes, identify questions, etc. The entire team should meet and discuss any questions you have noted, clarify a mutual understanding of requirements. However, don’t get bogged down into details – there will be time for that later.

1. Important: Setup a standing time to meet. At first it may be every day. Later you may move to every other day. Be specific about each team member’s assignment. Hold each other accountable.
2. For the project, you are following an Agile Software Development Process with one sprint. The User Stories are prioritized. The first 15 have the background shaded orangish, the next six are shaded greenish, and the final 11 are shaded yellowish. I would suggest thinking of these color bands as mini-sprints. In other words, start with the first 15. Identify:
3. Classes needed – quick class diagram by hand is recommended.
4. Design of Gui – quick sketch on paper is recommended.
5. Format of files for data persistence – This should be documented as it is critical in places.
6. System level tests (unit tests are written by the developer for the class/method). What exact tests will you run by hand to verify your user stories? What exact data will you use? This should be documented (Document 3).
7. Identify an initial set of tasks (not everything for the sprint) and enter them GitHub Project, and assign them to team members.

Note:

* A user story may be a single task (if estimated to be <= 4 hours), but more usually, it may be several or many tasks.
* Tasks may be to research something needed, e.g. GUI, or how to work with text files
* Tasks may be related to planning and specifying system tests. Ideally, this should be done immediately so that as a developer finishes a user story, it can be tested. And, as stated in the course, this ensures that the developer writes just enough code to pass the test.

1. Start working on tasks, move tasks on Project Board from *To Do* to *In Progress* and eventually to *Complete.*

# Grading Rubric

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Points** | |  |  |  |
| **Cat Wt** | **Num** | **Weight** | **Low** | **High** | **Criterion/Levels** | **Sample Scoring** | **Weighted Score** |
|  |  |  |  |  |  |  |  |
| **10%** | **Project Management** | | | |  |  |  |
|  | 1 | 5% |  |  | Use of Project Board in GitHub to manage the project | 75 | 3.8 |
|  |  |  | 90 | 100 | All project tasks are displayed as issues and in appropriate columns |  |  |
|  |  |  | 80 | 89 | Most project tasks are displayed as issues and in appropriate columns |  |  |
|  |  |  | 70 | 79 | Some project tasks are displayed as issues and in appropriate columns |  |  |
|  |  |  | 60 | 69 | Few project tasks are displayed as issues and in appropriate columns |  |  |
|  |  |  | 0 | 59 | Project board not used or minimally used |  |  |
|  |  |  |  |  |  |  |  |
|  | 2 | 5% |  |  | Use of User Stories spreadsheet to manage the status of project | 100 | 5.0 |
|  |  |  | 90 | 100 | All entries for the Code, System Tests, and Status columns are correct |  |  |
|  |  |  | 80 | 89 | Most entries for the Code, System Tests, and Status columns are correct |  |  |
|  |  |  | 70 | 79 | Some entries for the Code, System Tests, and Status columns are correct |  |  |
|  |  |  | 60 | 69 | Few entries for the Code, System Tests, and Status columns are correct |  |  |
|  |  |  | 0 | 59 | No entries for the Code, System Tests, and Status columns are correct |  |  |
|  |  |  |  |  |  |  |  |
| **20%** | **Testing** | |  |  |  |  |  |
|  | 3 | 10% |  |  | Appropriate unit tests | 85 | 8.5 |
|  |  |  | 90 | 100 | All methods that are required to, have unit tests |  |  |
|  |  |  | 80 | 89 | Most methods that are required to, have unit tests |  |  |
|  |  |  | 70 | 79 | Some methods that are required to, have unit tests |  |  |
|  |  |  | 60 | 69 | Few methods that are required to, have unit tests |  |  |
|  |  |  | 0 | 59 | No unit tests |  |  |
|  |  |  |  |  |  |  |  |
|  | 4 | 10% |  |  | Appropriate system tests | 75 | 7.5 |
|  |  |  | 90 | 100 | All implemented user stories have at least one system test |  |  |
|  |  |  | 80 | 89 | Most implemented user stories have at least one system test |  |  |
|  |  |  | 70 | 79 | Some implemented user stories have at least one system test |  |  |
|  |  |  | 60 | 69 | Few implemented user stories have at least one system test |  |  |
|  |  |  | 0 | 59 | No system tests |  |  |
|  |  |  |  |  |  |  |  |
| **30%** | **Implementation** | | |  |  |  |  |
|  | 5 | 5% |  |  | Design-Class Diagram | 73 | 3.7 |
|  |  |  | 90 | 100 | Correct UML Notation, accurately reflects code, design described in words correctly | |  |
|  |  |  | 60 | 89 | Some of these met: Correct UML Notation, accurately reflects code, design described in words correctly | | |
|  |  |  | 0 | 59 | No class diagram |  |  |
|  |  |  |  |  |  |  |  |
|  | 6 | 5% |  |  | Design | 73 | 3.7 |
|  |  |  | 90 | 100 | Correct MVC design |  |  |
|  |  |  | 80 | 89 | Mostly correct MVC design |  |  |
|  |  |  | 70 | 79 | Attempt at MVC design |  |  |
|  |  |  | 60 | 69 | No MVC, but some adherence to Single Responsibility |  |  |
|  |  |  | 0 | 59 | No coherent design |  |  |
|  |  |  |  |  |  |  |  |
|  | 7 | 15% |  |  | Implementation & Functionality of code | 85 | 12.8 |
|  |  |  | 90 | 100 | All implemented user stories run correctly & utilize best practices |  |  |
|  |  |  | 80 | 89 | Most implemented user stories run correctly & utilize best practices |  |  |
|  |  |  | 70 | 79 | Some implemented user stories run correctly & utilize best practices |  |  |
|  |  |  | 60 | 69 | Few implemented user stories run correctly & utilize best practices |  |  |
|  |  |  | 0 | 59 | No implemented user stories run correctly |  |  |
|  |  |  |  |  |  |  |  |
|  | 8 | 5% |  |  | Commits, Pull Requests & Code Review | 60 | 3.0 |
|  |  |  | 90 | 100 | Consistent & proper use of commits, Pull Requests & code review |  |  |
|  |  |  | 0 | 89 | Inconsistent &/or improper use of commits, Pull Requests, & code review |  |  |
|  |  |  |  |  |  |  |  |
| **40%** | **Individual Effort** | | |  |  |  |  |
|  | 9 | 35% |  |  | Individual Effort | 85 | 29.8 |
|  |  |  | 90 | 100 | 58+ quality hours logged for entire project |  |  |
|  |  |  | 80 | 89 | 47-57 quality hours logged for entire project |  |  |
|  |  |  | 70 | 79 | 37-46 quality hours logged for entire project |  |  |
|  |  |  | 60 | 69 | 40-49 quality hours logged for entire project |  |  |
|  |  |  | 0 | 59 | <40 quality hours logged for entire project |  |  |
|  |  |  |  |  |  |  |  |
|  | 10 | 5% |  |  | Individual Retrospective | 100 | 5.0 |
|  |  |  | 90 | 100 | Thoughtful, complete description, <= 2 grammar/spell errors |  |  |
|  |  |  | 80 | 89 | Thoughtful, complete description, >2 grammar/spell errors |  |  |
|  |  |  | 70 | 79 | Cursory description |  |  |
|  |  |  | 0 | 69 | Minimal description |  |  |
|  |  |  |  |  |  |  |  |
| **100%** | **Total** | **100%** |  |  |  |  | **82.6** |