COMP3111: Software Engineering

Conflict Resolution and Documentation

Learning Outcomes

- To resolve and prevent having merge conflicts
- To write basic Javadoc style comments and generate HTML-based documentation with Eclipse built-in tool

Exercise 1: Conflict Resolution

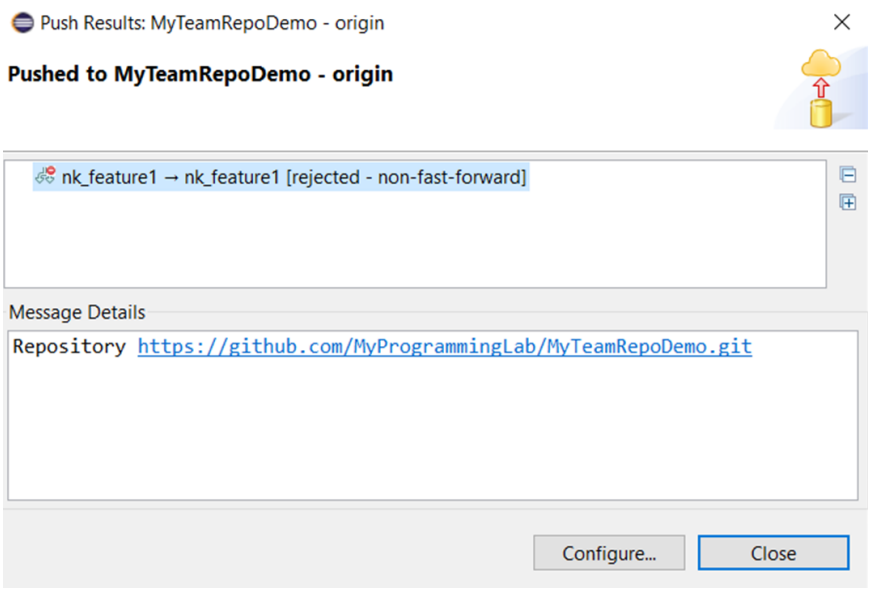
1.1: Typical workflow on software development with the team repo:

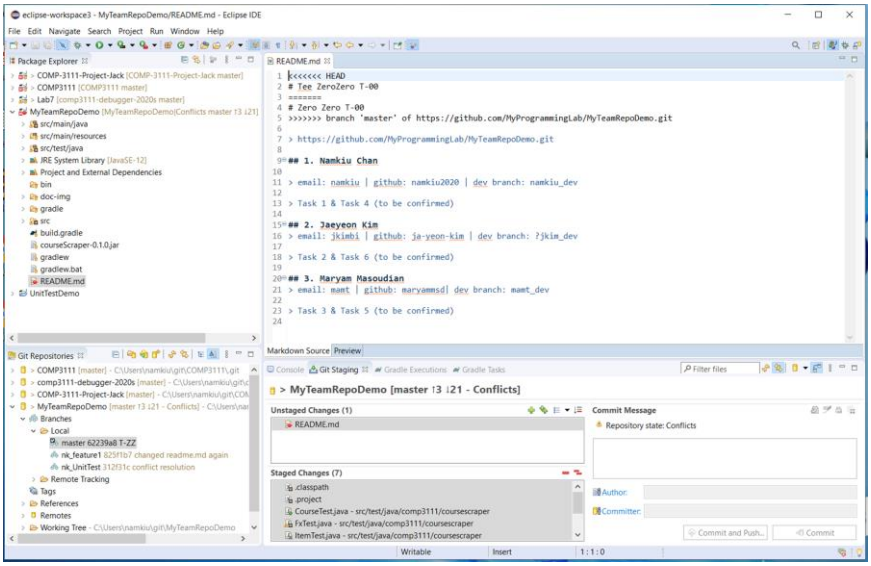
- At Eclipse, checkout an appropriate development branch
- Implement and conduct unit testing on a set of functionalities
- Make changes and commit to the local repo
- Pull, check, and merge from the team repo at GitHub [R1]
- Push the development branch to the team repo at GitHub
- At GitHub, create a Pull Request for merging into the master branch [R2]

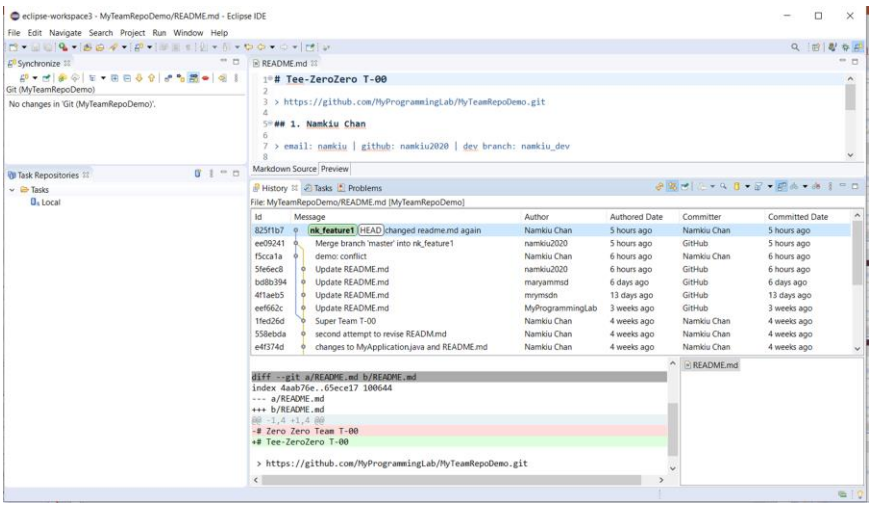
A merge request might result in conflicts which the developers have to fix manually. This is the case when the content of files cannot be merged automatically. Scenario illustrating the process of possible conflict resolution at different stages of [R1] and [R2] are provided at 1.2 and 1.3, respectively.

1.2: [R1] Resolving possible merge conflicts before pushing changes to the remote repo

A Sample Scenario: Screenshots

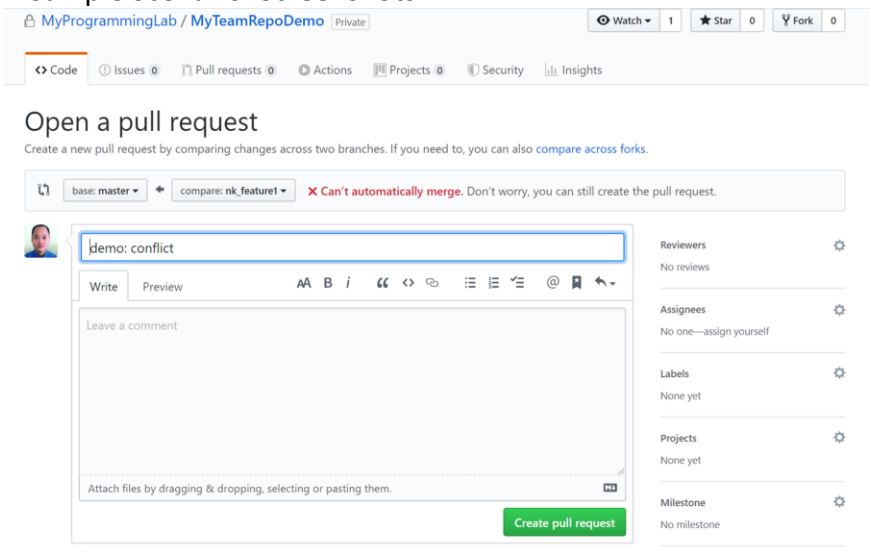


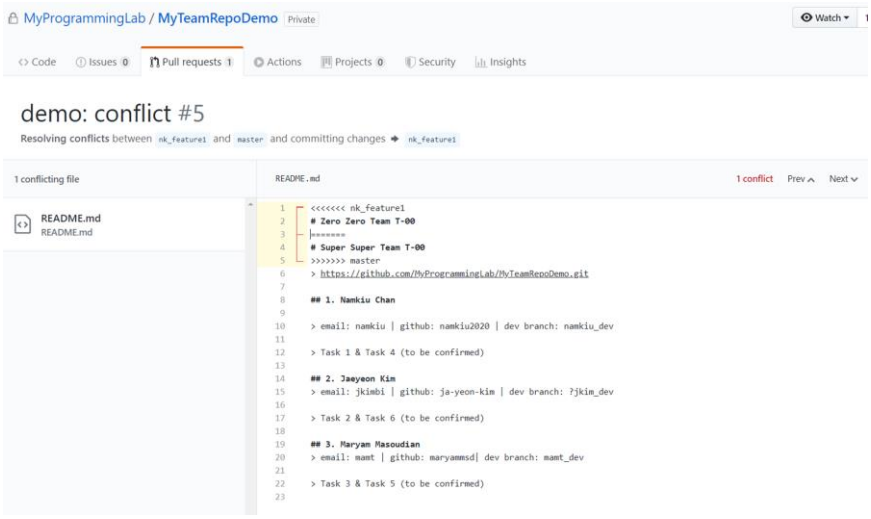


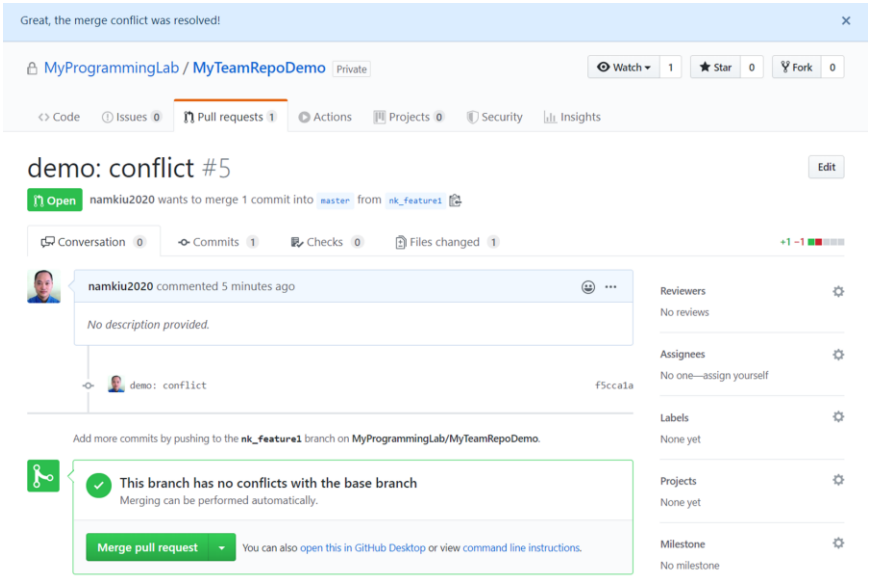


1.3: [R2] Resolving possible merge conflicts at the remote repo (handling Pull Request)

A Sample Scenario: Screenshots







Further Exploration: Introduce instances of merge conflicts to your team repo, and try resolving them (i) before pushing changes to the remote repo, and (ii) after submitting a Pull Request.

Exercise 2: Documentation

Javadoc is a tool for generating API documentation in HTML format from doc comments embedded in Java source code. Doc comments are special Java comments wrapped with /** ... */. A doc comment must precede a class, field, constructor or method declaration. All public and protected methods should be fully defined with Javadoc. Package and private methods do not have to be, but may benefit from it.

An Example:

```
/**
 * A program to say hello
 * @author     <a href=mailto:me@my.com>Me</a>
 * @version     1.0
 */
public class SayHello {
    public static void main(String args[]) {
        System.out.println("Hello, World");
        PrintHello("Charles");
```

```java
    }

    /**
     * @param name The name of the person we want to salute
     */
    public static void PrintHello(String name) {
        System.out.println("Hello, " + name);
    }
}
```

2.1: Run the Gradle Task “javadoc” under the category “documentation” in your project

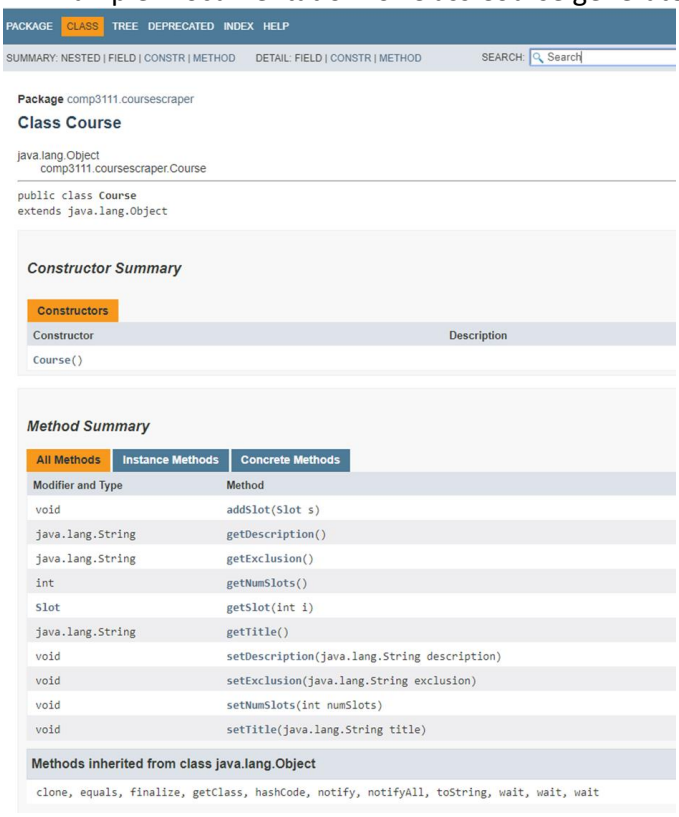
2.2: The HTML-based documentation generated can be found under the project workspace folder at: ‘build/docs/javadoc/index.html’

2.3: More on Javadoc

- Tutorial on Writing Javadoc Documentation
- Official Guide on Javadoc Tool

Further Exploration: Write Javadoc style comments for the classes you developed for your project and generate HTML-based documentation with the Eclipse built-in tool. For your project submission, you are required to generate Javadoc for the source files under src/main/java only. Javadoc for testing files under src/test/java can be excluded.

An Example: Documentation for Class Course generated from Javadoc



PACKAGE CLASS TREE DEPRECATED INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

Package comp3111.coursescraper

Class Course

java.lang.Object
comp3111.coursescraper.Course

public class Course
extends java.lang.Object

Constructor Summary

Constructors

Constructor	Description
Course()	

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method
void	addSlot(Slot s)
java.lang.String	getDescription()
java.lang.String	getExclusion()
int	getNumSlots()
Slot	getSlot(int i)
java.lang.String	getTitle()
void	setDescription(java.lang.String description)
void	setExclusion(java.lang.String exclusion)
void	setNumSlots(int numSlots)
void	setTitle(java.lang.String title)

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Lab Activity:

This is your final lab session for the course on software engineering. We invite you to consider what you have learned along the way. Here are the learning outcomes and target skills that we have introduced in the lab sessions as well as the team project briefing.

Skills-1: Developing Java applications (#eclipse #jdk #gradle)
Skills-2: Creating Java GUI applications (#javafx #scenebuilder)
Skills-3: Drawing class diagrams and use case diagrams (#drawio #datamodel)
Skills-4: Collaborative software development (#github #repo #commit #merge #pullrequest)
Skills-5: Debugging Java Code (#breakpoint #debugstack)
Skills-6: Unit Testing Java Code (#junit #jacoco #coverage)
Skills-7: Documenting Java Code (#javadoc)
Skills-8: Team Project – Data Explorer on COVID-19 (#teamwork #communications #requirements #coding #testing #documentation #algorithm #datamining #datavisualization)
Skills-9: Some unintended learning outcomes (#nicesurprises #wow)

Make a note of the learning outcomes that you think you have achieved, either fully or partially, regarding your skill learning journey on software engineering. What major problems have you encountered and how did you deal with them? What **specific skills** you utilized that allowed you to resolve the issues? Or, if there are ongoing challenges yet to be solved, what **specific skills** you aim at acquiring to resolve the issues?

Assessment

Submit through Canvas a single-page reflective essay (in PDF) addressing the following questions regarding your skill learning journey on software engineering:

- What major problems have you encountered and how did you deal with them?
- What **specific skills** you utilized that allowed you to resolve the issues?
- Or, if there are ongoing challenges yet to be cleared, what **specific skills** you aim at acquiring to resolve the issues?