CS2103T - Week 3 - Topics

W3.1 - RCS: Branching

W3.2 - RCS: Creating Pull Requests

W3.3 - Java

W3.4 - Code Quality: Coding Standards

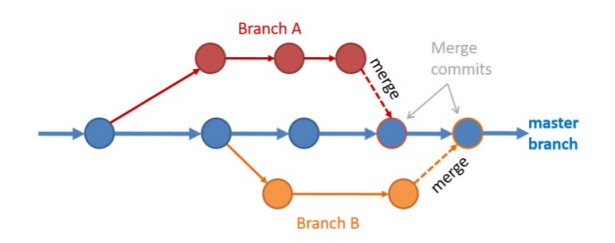
W3.5 - Developer Testing

W3.6 - Unit Testing

W3.1 — RCS : Branching

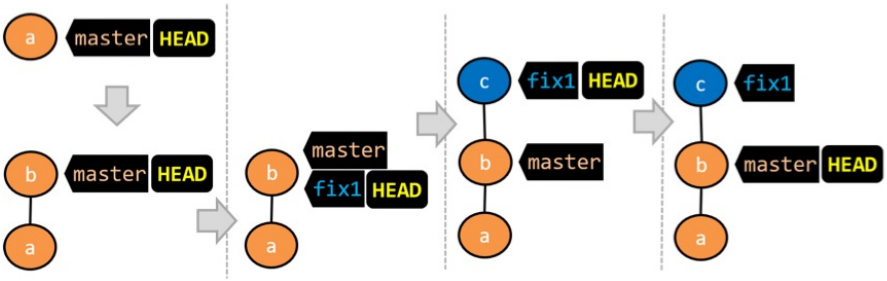
W3-1a - Project Management - Revision Control - Branching

- **Branching**: process of evolving multiple versions of the software **in parallel**
- A branch can be **merged** into another branch

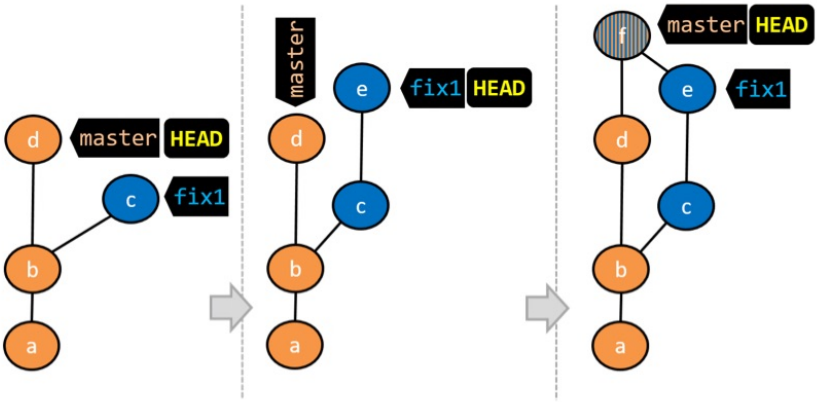


- **Merge conflicts** occur when trying to merge two branches that had changed the same part of the code and the RCS software cannot decide which changes to keep

W3.1b - Git and GitHub - branch : Doing multiple parallel changes



1. There is only one branch (i.e., `master`) and there is only one commit on it.
2. A new commit has been added. The `master` and the `HEAD` labels have moved to the new commit.
3. A new branch `fix1` has been added. The repo has switched to the new branch too (hence, the `HEAD` label is attached to the `fix1` branch).
4. A new commit (`c`) has been added. The current branch label `fix1` moves to the new commit, together with the `HEAD` label.
5. The repo has switched back to the `master` branch.



6. A new commit (`d`) has been added. The `master` label has moved to that commit.
7. The repo has switched back to the `fix1` branch and added a new commit (`e`) to it.
8. The repo has switched to the `master` branch and the `fix1` branch has been merged into the `master` branch, creating a *merge commit* `f`. The repo is currently on the `master` branch.

- merge vs rebase
- Git does a fast forward merge if possible, but in such cases, it is also possible to force Git to create a merge commit

3.1c - Tools - Git and GitHub - Dealing with merge conflicts

- Example :

```
COLORS
------
blue
<<<<<< HEAD
black
=======
green
>>>>>> fix1
red
white
```

W3.2 — RCS: Creating Pull Requests

W3.2a - Tools - Git and GitHub - Creating PRs

- Pull request (PR): mechanism for contributing code to a remote repo
- Repos must have shared history
- Upstream repo: repo you forked from
- PR review:

The next step of the PR life cycle is the PR *review*. The members of the repo that received your PR can now review your proposed changes.

- If they like the changes, they can *merge* the changes to their repo, which also closes the PR automatically.
- If they don't like it at all, they can simply close the PR too i.e., they reject your proposed change.
- In most cases, they will add comments to the PR to suggest further changes. When that happens, GitHub will notify you.

- Updating code will auto-update PR
- Sending PRs using separate branches is more common than master
- Possible to create PRs within the same repo

W3.3 - Java

W3.3a - Implementation - Documentation - Tools - JavaDoc - What

- Javadoc : tool for generating API documentation in HTML format from doc comments in source

W3.3b - Implementation - Documentation - Tools - JavaDoc - How

- Example for methods :

```java
/**
 * Returns lateral location of the specified position.
 * If the position is unset, NaN is returned.
 *
 * @param x  X coordinate of position.
 * @param y Y coordinate of position.
 * @param zone Zone of position.
 * @return Lateral location.
 * @throws IllegalArgumentException  If zone is <= 0.
 */
public double computeLocation(double x, double y, int zone)
    throws IllegalArgumentException {
  ...
}
```

- Example for classes :

```java
package ...

import ...

/**
 * Represents a location in a 2D space. A <code>Point</code> object corresponds to
 * a coordinate represented by two integers e.g., <code>3,6</code>
 */
public class Point{
    //...
}
```

W3.3c – C++ to Java – Miscellaneous Topics – File access

- Can use `java.io.File` class to represent a file object
- For Windows, backslash `\` has a special meaning,
  e.g. `"data\\fruits.txt"` or `"data/fruits.txt"`, NOT `"data\fruits.txt"`
- Can read from a file using a `Scanner` object that uses a `File` object as the source of data
- Can use `java.io.FileWriter` object to write to a file
  - Need to call `close()` of the FileWriter object for the writing operation to be completed
  - Can append also:

```
2       FileWriter fw = new FileWriter(filePath, true); // create a FileWriter in append mode
```

- `java.nio.file.Files` is a utility class that provides several useful file operations
  - Relies on `java.nio.file.Paths`

```
public static void main(String[] args) throws IOException{
    Files.copy(Paths.get("data/fruits.txt"), Paths.get("temp/fruits2.txt"));
    Files.delete(Paths.get("temp/fruits2.txt"));
}
```

W3.3d - C++ to Java - Miscellaneous Topics - Packages

To create a package, you put a package statement at the very top of every source file in that package. The package statement must be the first line in the source file and there can be no more than one package statement in each source file. Furthermore, **the package of a type should match the folder path of the source file.** Similarly, the compiler will put the `.class` files in a folder structure that matches the package names.

- Package names in all lower case, separated by dots
  - Packages in Java begin with java / javax

e.g.

```
1 package seedu.tojava.util;
```

- Importing public package members:

e.g.

```
3 import seedu.tojava.util.StringParser;
4 import seedu.tojava.frontend.*;
```

- No need to import to use package, just use its fully qualified name, e.g.

```
10          // Using the fully qualified name to access the Processor class
11          String status = seedu.tojava.logic.Processor.getStatus();
```

- Importing a package does not import its sub-packages
- Can use static import to import static members (fields, methods)

e.g.

```
1 import static seedu.tojava.util.Formatter.PREFIX; // constant
2 import static seedu.tojava.util.Formatter.format; // method
```

- When compiling / running Java, take package into account:

If the `seedu.tojava.Main` class in defined in the file `Main.java`,

- when compiling from the `<source folder>`, the command is:
  `javac seedu/tojava/Main.java`
- when running it from the `<compiler output folder>`, the command is:
  `java seedu.tojava.Main`

W3.3e - C++ to Java - Miscellaneous Topics - Using JAR files

- Java applications typically delivered as JAR (Java Archive) files
- Contains Java classes and other resources (icons, media files, etc.)
- e.g. `java -jar foo.jar` launches foo.jar

W3.4 - Code Quality: Coding Standards

W3.4a - Implementation - Code Quality - Introduction - What

- **Production code**: code being used in an actual product with actual users
  - Needs to be of high quality (duh!)

W3.4b - Implementation - Code Quality - Style - Introduction

- Aim of coding standard: make entire code base look like it was written by one person

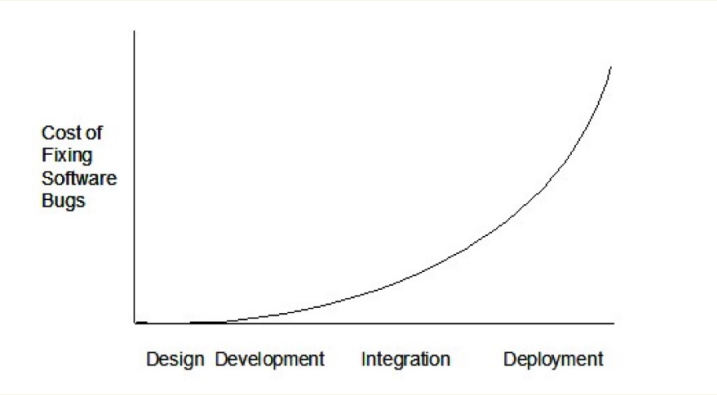
W3.5 - Developer Testing

W3.5a - Quality Assurance - Testing - Developer Testing - What

- **Developer testing**: testing done by **developers themselves** as opposed to professional testers / end-users

W3.5b - Quality Assurance - Testing - Developer Testing - Why

- Disadvantages of delaying testing until full product is complete:
  1. Locating the cause of such a test case failure is difficult due to large search space
  2. Fixing a bug found during such testing could result in major rework
  3. One bug might 'hide' other bugs
  4. Delivery may have to be delayed if too many bugs found
- The earlier a bug is found, the easier and cheaper it is to get it fixed



- Pros and cons of developers testing own code:

Pros:

- Can be done early (the earlier we find a bug, the cheaper it is to fix).
- Can be done at lower levels, for examples, at operation and class level (testers usually test the system at UI level).
- It is possible to do more thorough testing because developers know the expected external behavior as well as the internal structure of the component.
- It forces developers to take responsibility for their own work (they cannot claim that "testing is the job of the testers").

Cons:

- A developer may subconsciously test only situations that he knows to work (i.e. test it too 'gently').
- A developer may be blind to his own mistakes (if he did not consider a certain combination of input while writing code, it is possible for him to miss it again during testing).
- A developer may have misunderstood what the SUT is supposed to do in the first place.
- A developer may lack the testing expertise.

W3.6 - Unit Testing

W3.6a - Quality Assurance - Testing - Test Automation - Test automation using test drivers

- **Test driver**: code that 'drives' the SUT for the purpose of testing
  - Invoke SUT with test inputs and verifying the behaviour is as expected

W3.6b - Quality Assurance - Testing - Test Automation - Test automation tools

- JUnit for automated testing of Java programs

```java
@Test
public void testTotalSalary(){
    Payroll p = new Payroll();

    //test case 1
    p.setEmployees(new String[]{"E001", "E002"});
    assertEquals(6400, p.totalSalary());

    //test case 2
    p.setEmployees(new String[]{"E001"});
    assertEquals(2300, p.totalSalary());

    //more tests...
}
```

W3.6c - Quality Assurance - Testing - Unit Testing - What

- Unit testing: testing individual units (methods, classes, subsystems, ...) to ensure each piece works correctly
- In OOP, it is common to write one or more unit tests for each public method of a class

W3.6d - C++ to Java - JUnit - JUnit: Basic

- Foo → FooTest

Notes:

- Each test method is marked with a `@Test` annotation.
- Tests use `Assert.assertEquals(expected, actual)` methods to compare the expected output with the actual output. If they do not match, the test will fail. JUnit comes with other similar methods such as `Assert.assertNull` and `Assert.assertTrue`.
- Java code normally use camelCase for method names e.g., `testStringConversion` but when writing test methods, sometimes another convention is used: `whatIsBeingTested_descriptionOfTestInputs_expectedOutcome` e.g., `intDivision_zeroDivisor_exceptionThrown`
- There are several ways to verify the code throws the correct exception. The third test method in the example above shows one of the simpler methods. If the exception is thrown, it will be caught and further verified inside the `catch` block. But if it is not thrown as expected, the test will reach `Assert.fail()` line and will fail as a result.
- The easiest way to run JUnit tests is to do it via the IDE. For example, in Intellij you can right-click the folder containing test classes and choose 'Run all tests...'

W3.6e - Quality Assurance - Testing - Unit Testing - Stubs

- Unit testing should be done in isolation so bugs in the dependencies cannot influence the test
- Use stubs to achieve this

> **Stub**: A stub has the same interface as the component it replaces, but its implementation is so simple that it is unlikely to have any bugs. It mimics the responses of the component, but only for the a limited set of predetermined inputs. That is, it does not know how to respond to any other inputs. Typically, these mimicked responses are hard-coded in the stub rather than computed or retrieved from elsewhere, e.g. from a database.

- Other types of replacements: Mocks, Fakes, Dummies, Spies

W3.6f - C++ to Java - JUnit - JUnit : Intermediate