CS2103T – Software Engineering
AY24/25 Semester 2, Y2S2
Tutorials

Sim Ray En Ryan (A0273780X, e1123183@u.nus.edu)

December 3, 2025

Contents

1 Tutorial 1: MIPS Assembly Revision and Stack Frames

2 Tutorial 3

2.1 Discuss code quality of iP PRs

2.1.1 themintchoco's PR

Find instances of the three code quality problems listed below:

- Weak SLAP
- Nesting Problems
- Too-long Methods

2.1.2 Weak SLAP

In Parser.java, the parse method contains both high-level code (choosing which command), as well as some low-level commands (s.nextInt() - 1) that can either be extract as its repeated a few times, or left to the inner functions to deal with.

2.1.3 Nesting Problems

NA

2.1.4 Too-long Methods

Some of the class names seem unessacarily long, but the methods are ok.

2.2 User stories exercise

As a ...	**I can ...**	**So that I can ...**	**Notes**
first-time user	see some sample trips when I open the app	easily try out its features without needing to add my data first	
first-time user	see a help message explaining which features I should try first	start by trying features that are more suited for new users	e.g., "hey you seem to be new. Try adding a trip first"
new user ready to adopt the app for my own use	purge all data	get rid of sample/dummy data and start adding my real data	
busy user	track all trip-related data inside the app	save time looking for data	
user	sending trip info to friends	via email or telegram	

user	add a trip		
user	delete a trip	get rid of trip no longer needed to track	
user	edit trip details	correct mistakes I made when adding a trip	
user	view all trip details	recall details of trips	
user	see the next upcoming trip details when I open the app	save the step of searching for the trip	reason: the next upcoming trips is the most likely trip the user may want to see

Think of the answers to the following questions:

- Which user stories don't follow the correct formal?
- Any of them too big for the tP planning?
- Which are must-have features

2.3 Prioritize tP user stories

Find our user stories here.

2.4 Create a feature list for the MVP version

Based on the user stories, conceptualize the MVP in the form oa feature list. So far, we have user stories we want to include in the MVP version. But user stories simply tell us user needs. To move towards a product design, we need to design product features of the product can fulfill those user needs. Note down the feature list in your online project notes document.

3 Tutorial 4: Debugging and Sequence Diagrams

3.1 Debugging

- Set a breakpoint in iP
- Run the code in debug mode
- Step through 1-2 statements
- Take a screenshot of the IDE

3.2 Sequence Diagrams

3.2.1 Diagram 1: Machine

- m.producePrototype() is called
- prototype = new Unit()
- Loop the following 5 times
  - prototype.stressTest()
- Return the prototype

3.2.2 Diagram 2: ParserFactory

- A ParserFactory has been called for its createParser() method
- ParserFactory calls its own checkParams() method
- If it is not valid, continue, else
  - Create a Parser
  - Parser has a callBack to ParserFactory.getInfo()
  - If it is forward, ParserFactory calls Parser.set()
  - If it is backward, ParserFactory calls Parser.reset()
  - Then Parser is destroyed

4 Tutorial 5: PR Tracker

4.1 Exercise on Requirements 1

4.1.1 Write 1 must-have and 1 nice-to-have user stories, covering user types tutor and manager.

Must Have: As a tutor, I can find impending PRs from my mentees in one page so that I do not have to spend time enumerating through all mentees.

Nice to Have: As a manager, I can ping my tutors who have impending PRs from their mentees so that I can remind them to do their work.

4.1.2 Write at least 1 NFRs, related to performance/scalability and/or usability.

The Pull Request Tracker should be able to support up to one million total PRs, and not have any loading issues.

4.1.3 Give at least 1 terms worth recording in the glossary.

Managers: The people who manage the course, which includes the professor and the head TA

4.1.4 Complete the following use case. Give at least one extension. Note that the tutor should give comments in the order of PR size (i.e., give comments to smaller PRs first). Assume the following use cases exists already: U1. Sort PRs by a criterion, U2. Add comments to a PR

- System: PRT
- Use Case: U3. Add comments to mentee PRs
- Actor: Tutor
- Precondition: Mentee has send a PR
- 1. Sort PRs by size, ascending, using U1.
- 2. Get smallest PR, and add comments to it using U2.
- 3. Repeat step 2 until there are no more PRs.
- 2a. Have an additional option to have default PR comments for simple PRs.z
- 3a. Save PRs that have not been completed back into impending PRs.

4.2 Exercise on Requirements 2

Review a sample answer (provided by the tutor) for the task 1 above, to identify if it has bugs listed below, and discuss, as directed by the tutor.

4.3 Refine DG

After the tutorial, if applicable, refine the relevant sections of your own DG based on what you learned from the tutorial activities. Note: Ideally, you should have completed iteration v1.1 already. In that case, this DG refinement can be done as part of a subsequent iteration (e.g., v1.2)

5 Tutorial 6: UML Diagrams

5.1 Draw a class diagram and object diagram

Use triangle brackets for interfaces. Inheritance arrow should be a triangle and dotted line. bill should not appear. Java methods are by default public in a public class. Optional to have boxes.

5.2 Draw a sequence diagram

5.2.1 How would the diagram change if PersonList is updated

6 Tutorial 7: Code Coverage, CCDs, Activity Diagrams

6.1 Demo measuring code coverage

6.2 CCDs

UML Object Diagram

6.3 Activity Diagrams

7 Tutorial 8: Process Scheduling

7.1

8 Tutorial 9: Process Scheduling

8.1

9 Tutorial 10: Testing

9.1 Equivalence Paritions and Boundary Values

Design test cases for the `day` parameter

1. What are the equivalence partitions for the parameter `day` of this method?

```
/**
 * Returns true if the three values represent a valid day
 */
boolean isValidDay(int year, int month, int day) {
}
```

   They are 1 to 28, 29, 30, 31, and every other possible integer.

2. What are the boundary values for the parameter `day` in the question above?
   They are 0, 1, 28, 29, 30, 31, 32.

3. Give 10 test inputs you would use for the parameter `day` in the question above.
   They are 0, 1, 28, 29, 30, 31, 32, -1, 10, 20

9.2 Combining Multiple Test Inputs

Apply heuristics for combining multiple test inputs to improve the E&E of the following test cases, assuming all 6 values in the table need to be tested. Underlines indicate invalid values. Point out where the heuristics are contradicted and how to improve the test cases.

SUT: `consume(food, drink)`

Test case	food	drink
TC1	bread	water
TC2	rice	_lava
TC3	_rock	_acid

Each Valid Input at least once in positive test cases

Test case	food	drink
TC1	bread	water
TC2	rice	water

Invalid Inputs individually before combining

Test case	food	drink
TC1	_rock	water
TC2	rice	_lava
TC3	bread	_acid
TC4	_rock	_acid

9.3 Apply Design Patterns

Suppose the `Processor` component of a software application is designed as follows (refer to website).

If you want to provide the ability for other components to get notified when a `Job` is finished running, without the `Processor` component becoming dependent on those other components:

1. Which design pattern would you use?
2. Modify the above design accordingly.

Use the observer pattern. Processor will now have an internal list containing other components that should know that Job has finished running. When job finishes running, it will call the list.update(), updating the thing in the list with new information.

9.4 Patterns in the tP

9.4.1 Does AB3 use the MVC pattern?

Yes.

9.4.2 Does AB3 use the Observer pattern?

No.