Project v1.3

CS2901 Software Engineering I

1 Summary of Submissions

Submission	Minimum acceptable performance to consider as 'reached'
Updated User Guide	Follow Section 2.1
Updated Developer Guide	Follow Section 2.2
Add your project portfolio page	Follow Section 2.3
Issue tracker set up	As explained in Appendix B.1
v1.3 managed using GitHub features	As explained in Appendix B.2

1.1 Deadlines

1.1.1 Demo 1.3. Presentation: Monday, 20th May

You shoud consider into your presentation:

- Feature List.
- Show the Section Design of the DG

1.1.2 Submission Date: Monday, 20th May

The deadline for submiting all items in above Table is the midnight before Tuesday 21st May.

2 Documentation

Recommended procedure for updating docs:

- Divide among yourselves who will update which parts of the document(s).
- Update the team repo.
- Use markdown format for UG and DG documents.

2.1 User Guide (UG)

Start moving the content from your User Guide (drafts created in previous weeks) into the User Guide page in your repository. You should follow the below structure:

1. **Introduction.** Describe briefly your project.

2. **Requirements.** Enlist your functional and nonfuntional requirements.
   (a) Functional
   (b) Non-Functional
3. **Features.** Enlist your features.
   (a) Viewing help: `help`
   (b) Adding a person: `add`
   (c) etc ...
4. **FAQ**
5. **Command Summary** (Optional)
6. **Glossary**
7. **Anexo A.** Include your use stories and use cases.

2.2 Developer Guide (DG)

Similar to the User Guide, start moving the content from your Developer Guide (draft created in previous weeks) into the Developer Guide page in your team repository.

The main objective of DG is to explain the implementation to a future developer, but a hidden objective is to show evidence that you can document deeply-technical content using prose, examples, diagrams, code snippets, etc. appropriately. To that end, you may also describe features that you plan to implement in the future, even beyond v2.0 (hypothetically).

You should folllow the next structure:

1. **Introduction.** Similar to the User Guide, describe briefly your project.
2. **Features.** Enlist your features. For DG, the description can contain things such as:
   - How the feature is implemented.
   - Why it is implemented that way.
   - Alternatives considered.
3. **Design.**
   (a) **Architecture**
      - **Class Diagrams** describe the structure (but not the behavior) of an OOP solution. These are possibly the most often used diagrams in the industry and an indispensable tool for an OO programmer. Figure 1 shows an example of an architecture diagram.

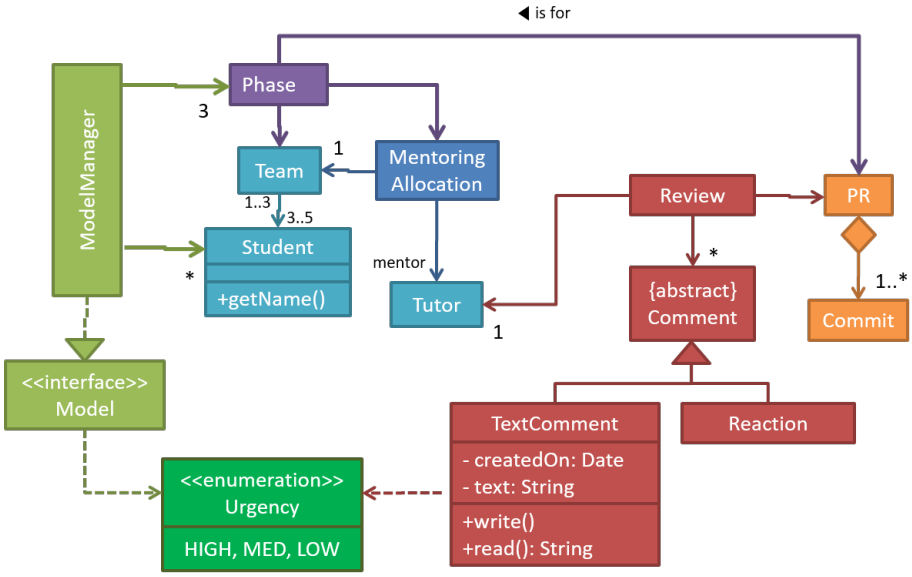


Fig. 1: An example of a class diagram

- **Architecture Diagram.**This diagram explains the high-level design of the App. The Architecture Diagram is a quick overview of each component. Figure 2 shows an example of an architecture diagram.

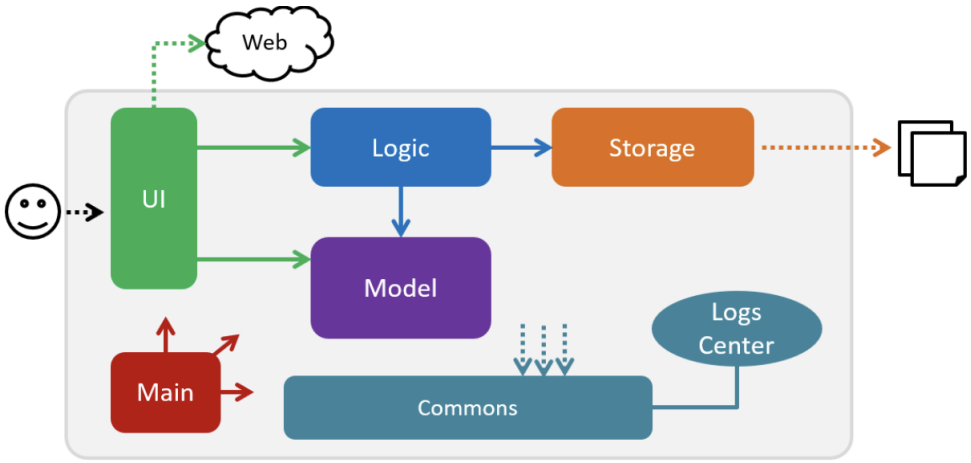


Fig. 2: An example of an architecture diagram

- **Component A.**
  i. Description of Component A.
  ii. A class diagram of Component A. Follow the example shown in Fig. 3.

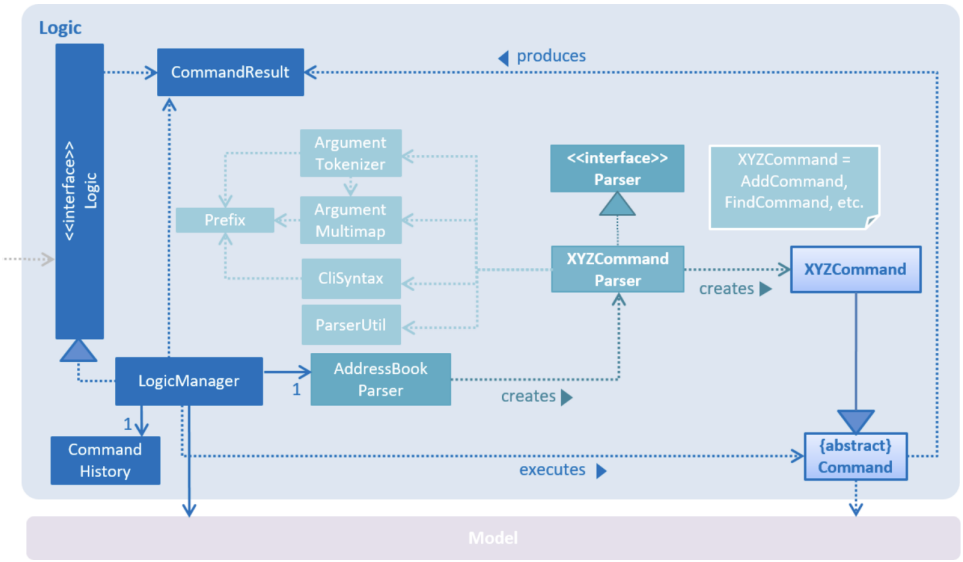


Fig. 3: An example of a class diagram of the Logic component

- **Interaction between Components.** In this section, you shoud add sequence diagrams that show how the components interact with each other for different scenarios (minimum 3 scenarios) of your project.

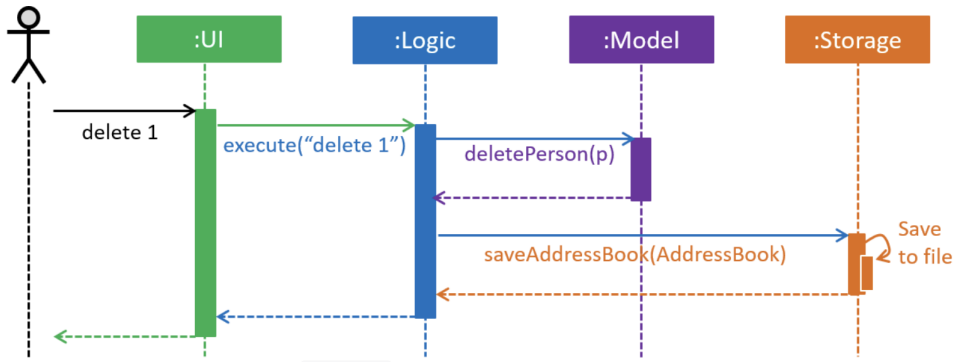


Fig. 4: An example of a sequence diagram for a particular command

4. **FAQ**
5. **Glossary**

2.3 Project Portfolio Page (PPP)

At the end of the project each student is required to submit a Project Portfolio Page. For this reason, you should start editing your PPP from now on.

The objective of PPP is:

- For you to use (e.g. in your resume) as a well-documented data point of your SE experience
- For us to use as a data point to evaluate your, contributions to the project and your documentation skills.

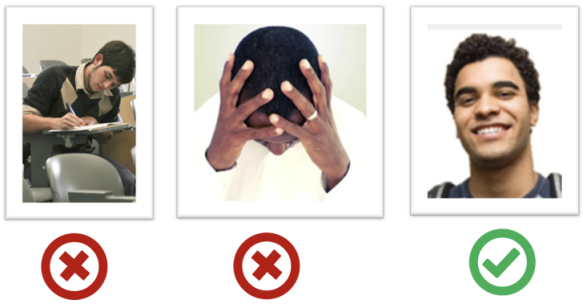
Consider the following sections:

- **Overview:** A short overview of your product to provide some context to the reader.
- **Summary of Contributions:**
  - Code contributed: Give a link to your code.
  - Features implemented: A summary of the features you implemented. If you implemented multiple features, you are recommended to indicate which one is the biggest feature.
  - Contributions to project management e.g., setting up project tools, managing releases, managing issue tracker etc.
  - Evidence of helping others e.g. responses you posted in our forum, bugs you reported in other team's products.
- **Contributions to the User Guide:** Reproduce the parts in the User Guide that you wrote. This can include features you implemented as well as features you propose to implement.The purpose of allowing you to include proposed features is to provide you more flexibility to show your documentation skills. e.g. you can bring in a proposed feature just to give you an opportunity to use a UML diagram type not used by the actual features.
- **Contributions to the Developer Guide:** Reproduce the parts in the Developer Guide that you wrote. Ensure there is enough content to evaluate your technical documentation skills and UML modelling skills. You can include descriptions of your design/implementations, possible alternatives, pros and cons of alternatives, etc.

3 Workflow

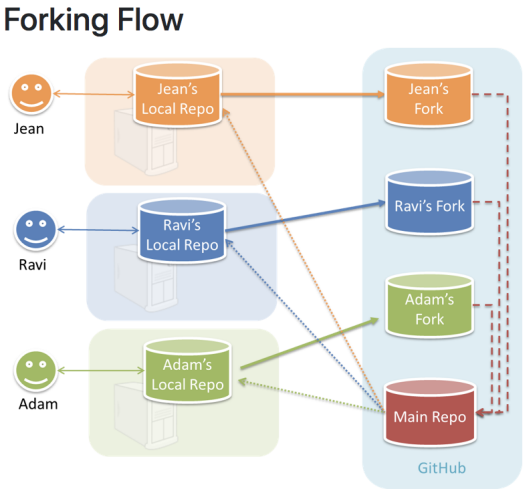
Before you do any coding for the project,

- Ensure you have set the Git username correctly in all Computers you use for coding.
- Use a suitable profile photo. The purpose is for the teaching team to identify you and professors to grade you. Therefore, you should choose a recent individual photo showing your face clearly (i.e., not too small) – somewhat similar to a passport photo. Given below are some examples of good and bad profile photos.



- Read our reuse policy (Appendix A), in particular, how to give credit when you reuse code from the Internet or classmates:

Follow **the forking workflow** (illustrated in below figure).

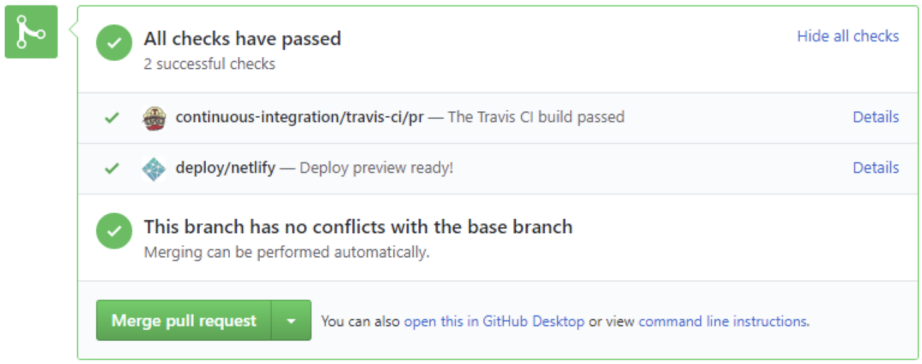


In particular, you should assure the following steps:

- Get team members to review PRs. A workflow without PR reviews is a risky workflow.
- Do not merge PRs failing CI . After setting up Travis, the CI status of a PR is reported at the bottom of the PR page. The screenshot below shows the status of a PR that is passing all CI checks.
- If there is a failure, you can click on the `Details` link in corresponding line to find out more about the failure. Once you figure out the cause of the failure, push the a fix to the PR.

A Policy on Reuse

Reuse is encouraged. However, note that reuse has its own costs (such as the learning curve, additional complexity, usage restrictions, and unknown



bugs). Furthermore, you will not be given credit for work done by others. Rather, you will be given credit for using work done by others.

- You are allowed to reuse work from your classmates, subject to following conditions:
  - The work has been published by us or the authors.
  - You clearly give credit to the original author(s).
- You are allowed to reuse work from external sources, subject to following conditions:
  - The work comes from a source of 'good standing' (such as an established open source project). This means you cannot reuse code written by an outside 'friend'.
  - You clearly give credit to the original author. Acknowledge use of third party resources clearly e.g. in the welcome message, splash screen (if any) or under the 'about' menu. If you are open about reuse, you are less likely to get into trouble if you unintentionally reused something copyrighted.
  - You do not violate the license under which the work has been released. Please do not use 3rd-party images/audio in your software unless they have been specifically released to be used freely. Just because you found it in the Internet does not mean it is free for reuse.
  - Always get permission from us before you reuse third-party libraries.

A.1 Giving credit for reused work

Given below are how to give credit for things you reuse from elsewhere.

If you reused code snippets found on the Internet e.g. from StackOverflow answers or referred code in another software

- If you read the code to understand the approach and implemented it yourself, mention it as a comment

```
//Solution below adapted from https://
    stackoverflow.com/a/16252290
{Your implementation of the reused solution here
    ...}
```

- If you copy-pasted a non-trivial code block (possibly with minor modifications renaming, layout changes, changes to comments, etc.), also mark the code block as reused code (using @@author tags). You can use the following format:

```
//@@author {yourGithubUsername}-reused
//{Info about the source...}

{Reused code (possibly with minor
    modifications) here ...}

//@@author
```

B Managing Issues

B.1 Issue Tracker Setup

We recommend you configure the issue tracker of the `main` repo as follows:

1. Add the following labels:
   - **Issue type labels:**
     - `type.Epic` : A big feature which can be broken down into smaller stories e.g. search
     - `type.Story` : A user story
     - `type.Enhancement`: An enhancement to an existing story
     - `type.Task` : Something that needs to be done, but not a story, bug, or an epic. e.g. Move testing code into a new folder)
     - `type.Bug` : A bug
   - **Status labels:**
     - `status.Ongoing` : The issue is currently being worked on. note: remove this label before closing an issue.
   - **Priority labels:**
     - `priority.High` : Must do
     - `priority.Medium` : Nice to have
     - `priority.Low` : Unlikely to do
   - **Bug Severity labels:**
     - `severity.Low` : A flaw that is unlikely to affect normal operations of the product. Appears only in very rare situations and causes a minor inconvenience only.
     - `severity.Medium` : A flaw that causes occasional inconvenience to some users but they can continue to use the product.

- `severity.High` : A flaw that affects most users and causes major problems for users. i.e., makes the product almost unusable for most users.

2. Create following milestones : `v1.0`, `v1.1`, `v1.2`, `v1.3`, `v1.4`.
3. You may configure other project settings as you wish. e.g. more labels, more milestones.

B.2 Using Issues

- **Define project tasks as issues.** When you start implementing a user story (or a feature), break it down to smaller tasks if necessary. Define reasonable sized, standalone tasks. Create issues for each of those tasks so that they can be tracked.e.g.

  1. A typical task should be able to done by one person, in a few hours.

  Bad: (reasons: not a one-person task, not small enough): Write the Developer Guide

  Good: Update class diagram in the Developer Guide for v1.4

  2. There is no need to break things into VERY small tasks. Keep them as big as possible, but they should be no bigger than what you are going to assign a single person to do within a week. eg.,

  Bad: Implementing parser (reason: too big).

  Good: Implementing parser support for adding of floating tasks

  3. Do not track things taken for granted. e.g., push code to repo should not be a task to track. In the example given under the previous point, it is taken for granted that the owner will also (a) test the code and (b) push to the repo when it is ready. Those two need not be tracked as separate tasks.
  4. Write a descriptive title for the issue. e.g. Add support for the 'undo' command to the parser

- **Assign tasks (i.e., issues) to the corresponding team members using the assignees field**. Normally, there should be some ongoing tasks and some pending tasks against each team member at any point.
- Optionally, you can use `status.ongoing` label to indicate issues currently ongoing.