Deliverables by iteration

Theodore S Norvell
Electrical and Computer Engineering
Memorial University

Due November 2, 16, 30

1 Iteration 0

Due November 2. Presentation November 2.

In your Git Repository. You should have a folder called: "requirements" in that folder you should have *first drafts* of the following documents, preferably in HTML format[1]

- Requirements document containing
  - A domain model (as one or more UML class diagrams.)
  - A lexicon – describing the terminology used in the project.
  - A list of requirements that you intend to implement by the end of term, broken into two sections:
    - Those you plan to implement in Iteration 1 (i.e. by November 16th)
    - Those you plan to implement in Iteration 2 (i.e., by November 30th)
- Specification document.
  - One of more Use Case diagrams.
  - A state diagram showing how the use-cases move the system from one state to another. In this diagram, there is one state for each precondition or postcondition used in the use-cases and each use case is represented by a transition that starts in the use case's precondition and ends in the use case's postcondition. (See the course notes for examples.)

[1] A good way to make HTML documents, if you don't want to use HTML directly, is to compose them with a "markdown editor" such as Moeditor or Typora. You can also export HTML from GoogleDoc or Word.

- Use cases. For Iteration 0, your specification document should contain use cases (ideally) covering (at least) all the user stories that you have planned for the first iteration. (Expanding the use cases to cover the Iteration 2 stories will be done in Iteration 1.)
- You might want to draw "wire frame" diagrams to illustrate the layout of the user interface. (Pencil Project is a nice tool for making wire frame diagrams.
- You might want to illustrate some of the use cases with sequence diagrams. In such a sequence diagram the system is represented by one lifeline and the messages represent user interface events (e.g. mouse actions and button clicks) and changes what the user sees.

2 Iteration 1

2.1 Documents and source code, due November 16.

1. The requirements documents should be revised. Indicate the state of each user story (done, started, not started). You may revise the list of user stories.
2. The specification document should be updated to include use cases that cover all user stories.
3. You should have a folder named design containing
   (a) High-level design document (in HTML).
      i. Containing one or more UML class diagrams showing your packages and the dependence relations between these packages. Also show library packages or third-party packages that are used.
      ii. For each of your packages, a short description of the purpose of the package.
      iii. For any third party packages, you use, explain the purpose of the package and where its documentation can be found.
      iv. For each of your packages, one or more class diagrams showing the classes and interfaces inside the package, associations between them and associations with classes in other packages that are depended on.
      v. For each class and interface, a description of the purpose of the class. I.e., what it represents or does.
   (b) A test plan that explains how you intend to test each package and class.
   (c) A short document that explains how to compile and execute your code.
4. You should have a folder named src (or source) that contains your source code.
   (a) Each class should be documented using JavaDoc (for Java), TSDoc (for Typescript), or Dox (for Haxe). Each public method should also be documented.

5. You should have a folder named tstsrc (or test-source) that contains your automated tests. For Java, please use JUnit 5 (or 4). For TypeScript, I recommend using Mocha. For Haxe, I've used Buddy with success. Your model, and controller should be tested. Also any classes in the view that it makes sense to test should be tested.

2.2 Presentation, November 16

You will again only have 10 minutes to present. You should concentrate on the high level design of your system.

3 Iteration 2

3.1 Documents and source code, due November 30.

1. The requirements documents should be revised. Indicate the state of each user story (done, started, not started).

2. The specification document should be updated to include use cases that cover all user stories.

3. You should have a folder named design containing

   (a) High-level design document (in HTML).

      i. Containing one or more UML class diagrams showing your packages and the dependence relations between these packages. Also show library packages or third-party packages that are used.

      ii. For each of your packages, a short description of the purpose of the package.

      iii. For any third party packages, you use, explain the purpose of the package and where its documentation can be found.

      iv. For each of your packages, one or more class diagrams showing the classes and interfaces inside the package, associations between them and associations with classes in other packages that are depended on.

      v. For each class and interface, a description of the purpose of the class. I.e., what it represents or does.

   (b) A test plan that explains how you intend to test each package and class.

   (c) A short document that explains how to compile and execute your code.

   (d) **NEW FOR 2!**. A short document that lists for each team member, which part they worked on.

4. You should have a folder named src (or source) that contains your source code.

(a) Each class should be documented using JavaDoc (for Java), TSDoc (for Typescript), or Dox (for Haxe). Each public method should also be documented.

5. You should have a folder named tstsrc (or test-source) that contains your automated tests. For Java, please use JUnit 5 (or 4). For TypeScript, I recommend using Mocha. For Haxe, I've used Buddy with success. Your model, and controller should be tested. Also any classes in the view that it makes sense to test should be tested.

3.2 Presentation, November 30 & December 1

You will have 12 minutes to present + 3 minutes for questions. You should demonstrate main the features of the system. You should discuss your design and how it promotes modularity, reusability, and extensibility. Each member of the team should speak during the presentation.