Project Release (Iteration #3) Assignment Handout

1. Introduction

Congratulations on completing the first two iterations of the semester project for CSci-3081W; now on to our final iteration, which we call our Project Release. The emphasis in this iteration will be on getting your project to the point where it is such a solid platform and code base that it is ready to be used by both end users of the software and other developers outside of your team. Writing is essential at this stage of software refinement – documentation, guidelines, tutorials, etc.

2. Requirements

Most of the technical requirements in this iteration revolve around writing, but we begin by describing an opportunity to fix any bugs that came up in previous iterations.

2.1 Fixing "Issues"

Here, as we come to our final iteration of the project this semester, we present an opportunity for you to go back and fix any bugs or incomplete features from prior iterations. If you lost points for these in earlier iterations, you can get those points back now if you log the problem as an Issue within your github.umn.edu repo, and then push a commit to resolve the issue. To get credit for this, here are the steps you should follow:

1. Enter an Issue into GitHub
2. Make a fix branch, referencing the Issue
3. Commit with reference to the Issue, following the template described in lecture
4. Merge the fix branch back into your working branch

More information can be found in the non-graded HW06 in the shared-upstream.

2.2 Writing to Support a "Project Release"

In this iteration, you should learn about and practice several complementary forms of writing for program design and development.

2.2.1 Code editing pass.

Do a code "editing pass" to go back over all of the code you have written for the project:

1. Add comments to describe the ***intent*** of key sections of code.

2. Revise variable and class names as needed to make them ***self-documenting***.

3. Make everything ***doxygen friendly***. All classes need descriptions. All public functions that are not getters/settings need descriptions. Some of the getters/setters might even need descriptions if not obvious. Anything that is not completely self-documenting should be documented.

2.2.2 Project Webpages.

Create a directory named "web" that is part of your project and make this the root for a project webpage that contains several things:

1. A main project page (index.html) that directs visitors to the different sections of the website. Distinguish on this main page between resources intended for users vs. resources for developers.

2. For users: A set of two tutorials with links to example data.
    a. "Getting Started with FlashPhoto" discusses brushes and filters.
    b. "Getting Started with Mia" discusses loading images and annotating them.

3. For developers: Major sections or links to sub-pages for the following topics.
    a. "Programming Reference": A link to documentation for all the code in the project (the library and the 2 applications) that you have auto-generated using doxygen.
    b. "High-Level Design": A discussion of the overall design of the project, using UML-like diagrams to illustrate major classes so that new developers can quickly understand the structure of the project.
    c. "Coding Style": A discussion of the most important rules of coding style that you enforce within the project. This should include conventions for naming variables, documenting code, use of namespaces, etc.
    d. "Common Tasks": Under this heading, include a tutorial written for other programmers that describes what they would need to do in order to add a new filter to the project.

3. Preliminary and Final Handins

To help you stay on track with the longer-term assignments you'll have in this course, we will break each assignment into multiple preliminary deadlines and one final deadline. You will be graded on the materials in your github repo master branch only at the time of the final deadline, but we strongly suggest that you work on the assignment a bit each week, pacing yourself according to the suggested timeline. At each preliminary deadline we will add some tests to the automated gitbot feedback scripts to give you a sense of how your current implementation matches the expectations we'll use for grading. At the final deadline, the feedback tests will be used are part of our automated grading, but we will also add some new tests that you do not get to see. The automated grading will make up just one portion of your final assignment grade. You'll also be graded on your writing, including design documents, diagrams, and code organization and style. More info on assessment is provided at the end of this document.

3.1 Iteration 3, Preliminary Handin (after ~1 week)

- **Resolve issues from prior iterations.**

If you plan to take advantage of the opportunity to fix any bugs or incomplete work from previous iterations, do this right away and give yourself a hard deadline of 1 week into the iteration to complete this work. You need to make sure you give yourself time to complete some pretty significant writing aspects of the assignment.

- **Learn / test doxygen.**

By the end of week one, also make sure you can reliably run doxygen on your code to auto-generate documentation and that you have learned the syntax for writing comments in the code that can be translated into documentation. Get all of these technical issues out of the way in week 1 so you can focus on content in week 2.

3.2 Iteration 3, Final Handin (after ~2 weeks)

- **Project webpages.**

Create the web pages described above, and pay special attention to the need to adjust your writing based on the audience, either users or fellow developers.

4. Grading

You can use this high-level grading rubric to better understand how we will grade the project and as a checklist to make sure you have completed everything.

- All parts of the web site (main page, two main section for users, four main sections for developers) are present.
- The major sections of the site are labeled and the "skimmability" is good, with either an easy way to jump between sections or major sections broken out as subpages.
- The writing takes the audience into account, adjusting the terminology, level of explanation, and content as appropriate for users vs. developers.
- UML diagrams, screen shots, and code snippets are integrated into the writing to make it more informative, concise, and clear.
- The discussion of high-level design draws upon knowledge of design patterns and good design practices discussed in class, conveying that you have successfully learned about these topics and understand how they relate to the project.
- The "coding style" section conveys stylistic ideas that agree with what we have discussed in class, and goes beyond saying simply, "we follow google style".
- The information is accurate (e.g., the descriptions of what it takes to add a new filter to the tool and of the high-level design).
- The doxygen-generated documentation is complete relative to the specs provided earlier in the document, specifically: All classes need descriptions. All public functions that are not getters/settings need descriptions. Some of the getters/setters might even need descriptions if not obvious. Anything that is not completely self-documenting should be documented.
- Inspection of the code reveals that it follows good self-documenting code practices for good variable names and that there are also cases of documenting the "intent" of important sections of code inline, that is, not just in situations that would be picked up by doxygen, but also in situations where other programmers reading the code in more detail would benefit.