

WattWaiter Oscar Web Application Code Review
By: Paul Galiza

A. Review the build

A1. Download the system, build it, and run any automated quality assurance checks (use "ant -f verify.build.xml"). If any errors occur, note these in your review.

The build was successful.

B. Review system usage

B1. Run the system. Exercise as much of its functionality as possible. Does it implement all of the required features?

The jar file works and all the function needed for this assignment is met.

B2. Try to break the system by providing it with unexpected input. Can you make the system crash or generate a stack dump? If so, note the input that caused the failure.

B3. Evaluate the user interface to the system. If it's a web app, is the interface self-explanatory? Does the system require unnecessary scrolling to see the data, or can it all fit in a single screen? Does it seem "professional", or does it look "amateurish"?

The web application looks professional. The whole thing was polished except that the resulting page after that date query was to huge. Just like what Prof. Johnson mentioned, a user should be able to see the whole result instead of scrolling for the rest.

C. Review the JavaDocs.

C1. Does the System Summary (provided in an overview.html file) provide an high-level description of the purpose of the system? Does it explain how each of the packages in the system relate to each other? Is the first sentence self-contained?

They have provided an overview summary.

C2. Do the Package Summaries (provided in package.html files) provide a high-level description of the purpose of the package? Do they explain how the classes in the package related to each other? Is the first sentence self-contained?

They have provided package summaries.

C3. Do the Class Summaries (provided at the top of each .java file) provide a high-level description of the purpose of the class? Does it provide sample code for clients of the class, if useful? Is the first sentence self-contained?

They have provided class summaries

C4. Do the Method Summaries (provided before each method) explain, from a client-perspective, what the method does? Do they avoid giving away internal implementation details that could change? Do they document any side-effects of the method invocation? (Note that you

can click on the method name to see the source code for the method, which is helpful to assessing the correctness and quality of the javadoc.)

They have provided method summaries and avoided giving away implementation details.

C5. Please review Chapter 4 of the Elements of Java Style for additional JavaDoc best practices, and check for compliance.

Some of the Class summaries start with “This Class” and they need better explanation of what each of their commands’ class does and they can do so with Elements of Java Style #47 and #50.

D. Review the names

D1. Do another pass through the JavaDocs, this time concentrating on the names of packages, classes, and methods. Are these names well chosen? Do they conform to the best practices in Elements of Java Style, Chapter 3? Can you propose better names?

They used meaningful names for their variable.

D2. Once you have reviewed the names displayed in the JavaDocs, review the source code for internal names in the same way.

They also used meaningful variables names within the source code.

E. Review the testing.

E1. Run Emma or some other code coverage tool on the system ("ant -f emma.build.xml"). Look at the uncovered code in order to understand one aspect of the limitations of the testing.

They did pretty good test cases and considering the uncovered code, they are mostly the exceptions for the each of the classes.

E2. Review the test cases. Is each component of the system exercised by a corresponding test class? Do the test cases exercise more than "happy path" behaviors?

The tests were in their own class and they were more than “happy paths” such as comparing the class generated result vs. the application generated result in ‘TestCarbonFlagSetter.

E3. Review the test output. Under default conditions, the test cases should not generate any output of their own. If output is desired for debugging purposes, it should be controlled by (for example) a System property.

Their outputs from the tests are controlled by the System property.

F. Review the package design

F1. Consider the set of packages in the system. Does this reflect a logical structure for the program? Are the contents of each package related to each other, or do some packages contain classes with widely divergent function? Can you think of a better package-level structure for the system?

The overall design of the system in terms of packaging was pretty good. They have separated the classes within packages that essentially does a single task.

G. Review the class design

G1. Examine its internal structure in terms of its instance variables and methods. Does the class accomplish a single, well-defined task? If not, suggest how it could be divided into two or more classes.

Their design did accomplish a single well-defined task.

G2. Are the set of instance variables appropriate for this class? If not, suggest a better way to organize its internal state.

The instances were appropriate since their design have each of the commands in separated classes.

G3. Does the class present a useful, but minimal interface to clients? In other words, are methods made private whenever possible? If not, which methods should be made private in order to improve the quality of the class interface to its clients?

From their design, I think that they should've made some of their commands private since the some of the because they don't concern each other.

H. Review the method design

H1. Does the method accomplish a single thing? If not, suggest how to divide it into two or more methods.

They have done a good job on reducing the commands to do a single task.

H2. Is the method simple and easy to understand? Is it overly long? Does it have an overly complicated internal structure (branching and looping)? If so, suggest how to refactor it into a more simple design.

The methods are generally straightforward and easy to understand thanks to their inline comment that tells exactly their loop does and the different recurring effects.

H3. Does the method have a large number of side-effects? (Side effects are when the result of the method's operation is not reflected purely in its return value. Methods have side-effects when they alter the external environment through changing instance variables or other system state. All "void" methods express the results of their computation purely through side-effect.) In general, systems in which most methods have few or zero side-effects are easier to test, understand, and enhance. If a method has a large number of side-effects, try to think about ways to reduce them. (Note that this may involve a major redesign of the system in some cases.)

Their methods don't have side-effects since they are returning actual values.

I. Check for common look and feel

I1. Is the code implemented consistently throughout the system, or do different sections look like they were implemented by different people? If so, provide examples of places with inconsistencies.

The overall look and feel of the code is implemented consistently throughout the system.

J. Review the documentation

J1. Does the project home page provide a concise and informative summary of what the system accomplishes? Does it provide a screen dump that illustrates what the system does?

They have included a good summary and they have included a screenshot of the web application.

J2. Is there a UserGuide wiki page? Does it explain how to download an executable binary of the system? Does it explain the major functionality of the system and how to obtain it? Are there screen images when appropriate to guide use? Try following the instructions: do they work?

Their user guide provided the necessary steps and information to run this application.

J3. Is there a DeveloperGuide wiki page? Does it explain how to download the sources and build the system? Does it provide guidance on how to extend the system? Try following the instructions: do they work?

They have provided good information for developers. One of the things I like is how they included prerequisite information so that developers know exactly what they need to run this application.

K. Review the Software ICU data

K1. Is the Software ICU gathering data consistently and reliably? If not, what data appears to be missing, and why might it be missing?

Their ICU data looks consistent and looks stable.

K2. If data is present, what does it tell you about the quality of the source code? Is the current level high, medium, or low? What are the trends in quality?

The coverage is high, complexity and coupling low and consistent. Churn shows that they started working early, development time and commit are consistent and they have some daily builds.

K3. If data is present, what does it tell you about the group process? Are all members contributing equally and consistently?

According to the ICU telemetry, two of the three members mostly worked on this project since the last member on committed ones.

L. Review Issue management

L1. Does the issue management page indicate that the project members are doing planning? In other words, that they have broken down the project into a reasonable set of issues (i.e tasks taking one day or longer are represented by an issue)?

They didn't really post any issues. I'm guessing they mostly delegated jobs during meetings since the only issue they posted is to "Read the book."

L2. Does the issue management page indicate that each member is contributing to the project? Does each member have at least one current "open" task? Has each member completed a reasonable number of tasks?

I can't really tell since there aren't any important issues posted.

M. Review continuous integration

M1. Is the system being built regularly due to commits? Are there long periods (i.e. longer than a day) when there are no commits?

They have consistent commit builds and daily builds.

M2. If the system build fails, are there situations in which the system stayed in a failed state for a long period of time (i.e. longer than an hour)?

No they have tried to fixed failed builds right away.

M3. Is there a continuous integration job? If so, check the configuration. Is it correct? Will it be triggered by a commit? Check the console output from one of the invocations. Are the appropriate Ant tasks (i.e. checkstyle, pmd, junit, etc.) being executed?

They have continuous integration jobs being executed.

M4. Is there a daily build job? If so, check the configuration. Is it correct? Will the job be run regularly once a day? Check the console output from one of the invocations. Are the appropriate Ant tasks for a daily build job (i.e. coverage, coupling, complexity, size, etc.) being executed?

They have daily build jobs and is running the right tasks.

Summary

Based upon the above sections, provide an overall summary of your perspective on the project. Does it fulfill the requirements, and if not, why not? Is the code of reasonable quality based upon its design, implementation, and quality assurance techniques? If not, what should be improved? Does the group appear to be functioning well based upon Software ICU, Issue, and Continuous Integration data? If not, what appears to be wrong and what should they do to improve?

My overall feel of the application is that it was designed, both the program and the application interface, very good so much so that it could almost be passed as professionally built. The negative things I have to say are mostly nitpicks that this checklist looks for. Other than that it was a very robust application.