Heuristic Evaluation

Andrea Lee
Team 6: Code Critters

Design

The Code Critiquer has a simple tool to help users find “antipatterns” common mistakes and poor habits made by novices. The user interface for the Code Critiquer includes a window containing a program provided by the user with a number assigned for each line of code. In addition to manually typing code, users are able to upload program files into the application. If the user creates an account and signs into the application, they will be able to view past critiques saved by the app itself. Critiques can also be downloaded.

UI Domain

The Code Critiquer is a web application, a program meant to be accessible through an internet browser rather than being downloaded separately. This means that it must be compatible with a variety of different browsers and operating systems. It should have the same formatting regardless of what system it is being used on. As it also has a “login” function, there must be a clear distinction between when a user is logged in or logged out.

This web application is also meant to be a programming tool. Lines of code are generally assigned specific numbers with any reference to these lines of code using these numbers. The majority of the application space should be dedicated to the code provided by the user.

Principles

This evaluation will be using a slightly modified version of Norman’s 10 Usability Heuristics.

1. Visibility of status – Users should know the effects of their actions.
2. Understandability of system status – If the different states of the system are visible, the user must also be able to distinguish them
3. User control – Users should be allowed to alter certain parameters to better achieve their goals
4. Control consistency – Each of the system's controls should function in predictable ways.
5. Prevent errors – The system should include failsafes to prevent users from making mistakes before they happen.
6. Recognizability – The user should be able to navigate the interface based on terms they should already be familiar with.
7. Efficiency – The users should be able to reach their intended goals in the simplest manner possible.
8. Aesthetics – No unnecessary information should be present.
9. Error recovery – If an error cannot be prevented, a user should have the opportunity to discover and correct it on their own.

10. Documentation – Information about the system's functions should be available to users.

Problems

The command for running the code is labeled “Check for Antipatterns”. There is no guarantee that all users will know what antipatterns are. This omission violates the “recognizability” and “documentation” heuristics. This command should either be rephrased or a description of antipatterns should be available somewhere within the app.

At the end of the example antipattern text, a number is given at the end. There is no information available anywhere on the app to notify users what this number means, violating the “recognisability”, “documentation” and “understandability of system status”. Users should be given some method of learning what these numbers mean.

The window for selecting files to upload contains a blank space above the file name field and upload button. Removing or filling this space would improve the “efficiency” and “aesthetics” heuristics.

Critical Concerns

While there is a clear distinction between the states of a logged in user and a guest, there is no distinction between what programming language is currently being used by the application. There is also no clear way to select the programming language either This violates the “visibility of status” and “user control” heuristics. Different languages will have different antipatterns and a mistake in one language can be standard practice in another.

For example, while semicolons are necessary to mark the end of every line in the C and Java programming languages, they are optional in the Python programming language. If Jane Doe manually types a Java program while the system is checking for Python antipatterns, the system will not detect a missing semicolon that would prevent the program from running properly.

Rather than saving past work with the name of the file itself, the code critiquer appears to only use the date and time it was uploaded. Therefore, if a user wishes to view a specific program's critiques, they will need to remember the exact date and time it was uploaded. As the user continues to use the code critiquer, this problem will only worsen. There is not an efficient way for the user to search through past critiques to find a specific program.

An example of this design working against the user would be after a semester of using the program, Jane Doe wants to look though the specific critiques for a specific

program that will be relevant for an upcoming exam. However, while she remembers the name of the program, she does not remember the exact date or time of the version she is looking for. To find this critique, she may need to look through every saved critique until she finds the one she needs. Program names should be given when browsing past critiques.