Amy Selsor
Dr. Page
Software Engineering I
15 November 2011

Programming Environment Survey

1. A good programming environment should provide:
    1. A layout that will provide the programmer to change the size of the source code editing field as well as be able to view and compare multiple files of code next to each other.
    2. A compiler and debugger that will not only run the code provided, but return comprehensive and understandable error messages when any defects are found in the code itself while running. This includes the line number, and what needs to be fixed within the code on that line or in that section.
    3. An automatic formatting (ie. indentation and parenthesis checker) style. A programming environment should automatically format the code into the default format for that certain language. This not only helps keep the code organized, but saves time for the programmer.

2. The first thing that I liked about the Dracula programming environment was the ability to check syntax using the "Check Syntax" button next to "Run". It helped prevent syntactical errors that could occur when trying to run the program. Another good aspect of Dracula includes the ability to produce the PSP++ reports. Even though it took up programming time to record the defect and time logs, the report actually helped me on a few occasions when I could look back at the logs and find how to fix a defect. Something that may have taken me 15 minutes previously to correct now takes only two minutes because of the record. My favorite part of Dracula was the ability to test functions individually in the console as well as in test files. If there came a point where I was unsure of how a function would work in certain situations, it was fairly easy to go to the console and test it out without disturbing the rest of the code.

3. There are a few aspects of the Dracula programming environment that I did not agree with. The first has to do with the layout of the program. As a programmer, I like to program in full screen. With Dracula, it severely limited the amount of space that was provided to the user because of the side bars for theorem checking. There should be an option to minimize that section so that the code can take up most of the screen. Secondly, I didn't like how I had to switch between windows of separate files that were stacked on top of each other on the computer screen. This problem would be fixed if there were some sort of tab layout so that the user could either switch between the files easily, or lay them side-by-side. Lastly, I had the program shut down on me on occasion when I was in the middle of programming. This may be an isolated incident that occurs only on my system, but there were certain times when the typing was delayed and I had to wait for the program to boot back up to keep going. It usually happened when I had multiple

To: Dr. Page

From: Amy Selsor

Re: Programming Environment to be used for nuclear reactor : Dracula

It has been decided that Dracula will be the programming environment used in the upcoming nuclear reactor project. Because of the implications of building a nuclear reactor in a highly populated area, extra precautions need to be taken in order to ensure the town inhabitant's safety. The best way to do so is to have ample amount of testing done on the code to make sure that the functions that are being used on the reactor are correct and have a high

Dracula helps out the programmer in checking the validity of the functions that are created by using various forms of testing and theorem proofs. One of the main forms of testing that is built into Dracula is a randomized-testing tool called DoubleCheck. DoubleCheck takes *defproperty* functions that are created by the programmer and tests them using random values, whether the function calls for integers, strings, boolean, etc. It will take the property and its random values and check them against the function in the file to make sure that the function has a high success rate. DoubleCheck will show which random values succeed and which values fail, giving the programmer the opportunity to look at the code and determine if anything needs to be corrected or changed. This is an invaluable tool considering the time it would take for a programmer to test 50-100 random values by him/herself.

Another form of testing that can be used incorporates the keyword *check-expect.* This allows the programmer to check the function against an expected value. This is not randomized-testing, but still allows the programmer freedom to make sure certain functions are working by creating properties that should succeed. For example, if in the nuclear reactor program, we designed the plant to shut off power sources when the reactors reach a certain temperature, we can test this individual temperature and make sure it returns the expected outcome.

Theorem proofs are another built-in functionality that Dracula provides. Theorem proofs allow the programmer to use *defthm* to prove theorems that contain previously defined functions in the file. Nuclear reactors involve all kinds of theorems, especially those stemming from thermodynamics as well as chemistry. It is important to make sure that these theorems are programmed correctly in order for the reactor to have the characteristics and reactions we want it to. The theorem proofs can help one with such proofs and will also point out the sections where the proofs may fail, allowing one to reformulate or built exceptions.

In conclusion, the primary reason for choosing Dracula over any other programming environment for the control software for the nuclear reactor is that Dracula provides built-in resources for testing code. This testing framework will allow for the highest rate of success for the project.