ID-10-T

Software Metrics Calculation System

Final Evaluation Report

Version 1.0

Prepared by:
Christopher Silva
Shujing Zhang
Anthony Enem
Nathan Durst
Da Dong

Contents

1 Description

The main objective of SMCS is to help first and second year computer science students become better programmers. This software will calculate and display metrics about the users source code such as lines of code, lines of documentation, cyclical complexity, etc. These metrics will be used to give the user feedback on their code and to correct commonly made mistakes.

2 Product Pros and Cons

The following is a list of positive and negative aspects of the SMCS:

2.1 Pros

- Friendly user interface
- Low learning curve
- Produces highlighted syntax of input code
- Allows for selection of metrics rather than listing all of them

2.2 Cons

- Supports C++, Java, and C only
- In some cases, produces incorrect results for block comments due to lack of negative look-ahead regex
- Professors don't have a way of seeing the report without getting it from the user
- Doesn't calculate the cyclomatic complexity metric

3 Life Cycle Model

We used the Waterfall Life Cycle Model to complete our project. We worked on each phase at a time and didn't start a new phase until we had finished the previous one. We started with requirements and design. Once they were completed we moved on to implementation. Finally, we finished the life cycle with testing, deployment and maintenance. The Waterfall Life Cycle allowed us to build off of our previous components which worked well with organization and distribution of work.

4 Team Organization

For our team structure, we mostly used a voluntary system but at some points team members were appointed tasks by the group leader. Drafts of documents were completed by individuals and final revisions were done as a team. Also, since our system supports 3 programming languages, we split the implementation into 3 separate groups to decrease implementation time.

5 Different Approach

If we were given a second chance, we would like to have met more frequently, however, due to scheduling conflicts we were unable to meet more than twice a week. If we had more time we would have liked to spend this extra time preparing for the testing phase. We also would like to have implemented a more interactive user interface and maybe include more metrics and support more languages such as C# or Python.

6 Future Enhancements/Improvements

In future iterations, we would like to:

- Include more metrics
- Support more OOP languages like C# and/or Python
- Improve the user interface to be more interactive and appealing
- Provide feedback messages to user to suggest improvements that can be made to their code
- Send a detailed report to the users professor

7 Conclusion

As a team, we learned:

- How to work and cooperate as a team, and how to delegate the work so that it was distributed evenly.
- Enough is good enough. Its important that you don't spend too much time on requirements or implementation that you leave little or no time for the testing phase which takes the most time.
- Always allow extra time for testing, because testing is crucial.
- Not to reinvent the wheel. Meaning, there are tons of other add-ons or tools that can be used instead of spending time implementing your own way.
- How the different phases of the development life cycle work in relation to one another and which ones take the most time.
- Follow the principles that relate to the life cycle model, such as Agile or Waterfall, so work will be more efficient.