Supporting Variation in Software through IDE Colorization

By
Miles Van de Wetering

A THESIS

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented June 05, 2017
Commencement June 2017

AN ABSTRACT OF THE THESIS OF

Miles Van de Wetering for the degree of Honors Baccalaureate of Science in Computer Science presented on June 05, 2017. Title:
Supporting Variation in Software through IDE Colorization

Abstract approved:

____

Eric Walkingshaw

This is the abstract for my honors thesis.

Key Words: variation, software product line, interactive development environment

Corresponding e-mail address: vandewmi@oregonstate.edu

Supporting Variation in Software through IDE Colorization

By
Miles Van de Wetering

A THESIS

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented June 05, 2017
Commencement June 2017

Honors Baccalaureate of Science in Computer Science project of Miles Van de Wetering presented on June 05, 2017

APPROVED:

Eric Walkingshaw, Mentor, representing Computer Science

Committee Member Name, Committee Member, representing Committee Member Department

Committee Member Name, Committee Member, representing Committee Member Department

Toni Doolen, Dean, Oregon State University Honors College

Miles Van de Wetering, Author

Contents

1 The first Section

Some text in section one.

1.1 A subheading

Some text under the subheading.

2 Another Section

Section text.

2.1 Our findings

Findings summary.

2.1.1 More information

Some more information.

3 Conclusion

Conclusion text.

4 Implementation

We hypothesize that complicated preprocessor syntax like #ifdef can be made more understandable to the programmer through careful use of colorization combined with relatively familiar features like code folding. Ideally, these features would be integrated with an Integrated Development Environment (IDE) in order to allow the programmer to leverage her usual workflow alongside the new tools at her disposal. For the first several months of work on this project, my time was devoted to selecting and evaluating different IDEs that might be useful for our purposes. Unfortunately many of the IDEs that I experimented with suffered from the same problem while they allowed adding new language highlighting rules, they generally did not allow the basic functionality of the editor to be modified. Two that I considered carefully were the JetBrains IDE and the ____ IDE (tree editing). JetBrains was eventually discarded because it did not allow me the versatility and control that I required in order to make extensive modifications. While the ____ IDE did allow for major changes to its functionality, we determined that it was too obscure to be of any great use for a user study. We wanted to select a popular, simple IDE so that we could evaluate its impacts as accurately as possible on our sample population.

Eventually Atom was selected. Atom is a highly configurable development environment built on the Chrome web browser. This meant that all modifications would be written in JavaScript. We wrote a parsing program in Haskell using the Megaparsec library. Haskell was chosen as the language for the parser because other members of our lab are already very familiar with the language. This made it more maintainable for our group. The Haskell parser reads in a C or C++ file and constructs a tree structure that defines all information required for the Atom plugin to replace concrete #ifdef syntax with colors. The structure was defined as follows:

Document: list of Nodes

Region: list of Nodes

Node: either a Choice or Text

Choice: A left Region, and an optional right Region

The parser returns a document to the JavaScript code whenever the plugin is toggled. Using this meta-syntax tree that the parser has constructed, the JavaScript removes the #ifdef syntax and replaces it with colors. Atom allows for ranges of text in the editor to be marked, which was extremely useful because this meant that changes the user makes may be saved directly into the same parts of the tree that the original text came from.

During the development process, complexity of the plugin grew significantly as features were added and edge cases discovered. Eventually this became almost unmanageable in JavaScript, so the code base was converted to Typescript (semantically equivalent to JavaScript, but allows for typed variables and a better object model. This made code maintenance and debugging much easier. The plugin allows users to select which colors will represent each dimension of variation. It also allows them to filter the code to see what will be compiled if different dimensions are defined or undefined. Code that is hidden with this feature may be viewed with a mouse-over tooltip. It allows the creation of new dimensions, and the insertion and deletion of arbitrarily nested choice nodes. Undo was never implemented in the plugin, unfortunately, because keeping track of a stack of tree states was not worth the usability gains for the purpose of our study.

In the future, the ability to undo edits in the plugin would be extremely useful. To complete the project, four variants of the plugin were made and published. The first is simply the default Atom IDE with undo disabled (undo was disabled in every version of the plugin). This was necessary to ensure that all treatment groups had access to the same basic features to avoid skewing results. The second included colorization and removal of #ifdef syntax, but did not allow variant folding. The third allowed variant folding but did not colorize the code, nor did it remove any of the #ifdef syntax. The fourth included all features.