HOMEWORK 6

Cody Croletto (ccrolett)
Laurel Williams (lwillia)

EECS 481 -- Software Engineering

NAME AND EMAIL IDS

Cody Croletto	ccrolett@umich.edu
Laurel Williams	lwillia@umich.edu

SELECTED PROJECT

We chose Microsoft's Visual Studio Code (https://github.com/Microsoft/vscode), an open source text editor that is meant to be entirely independent of the company's other IDE, Visual Studio. The text editor includes its own debugger and a dedicated terminal in hopes of creating a harmonious unity of the quick and easy-to-use functionality of a text editor and the core capabilities of a debugger. It is intended to be lightweight, with the option to install extensions to allow for more functionality.

PROJECT CONTEXT

VS Code was developed as an alternative to Visual Studio, which was only available on Windows, so Microsoft wanted a cross-platform option. It is not intended to be a full IDE, but instead, a lightweight code editor. It is similar to text editors like Sublime and Atom. VS Code is free, open-source and available on macOS, Linux and Windows. VS Code distinguishes itself from other text editors with its dedicated, fully operational terminal and its debugging mode, complete with breakpoint functionality.

From a business standpoint, Microsoft does not directly earn money from VS Code. The goal was to encourage developers to use more of their products, such as their Azure cloud services and their full-blown IDE, Visual Studio. They hope to garner customer satisfaction with their products and engage with more developers. A positive experience with VS Code could result in developers choosing other Microsoft products.

PROJECT GOVERNANCE

Visual Studio Code is made publically available on GitHub for users to contribute. Contributors communicate almost exclusively on GitHub. For a user requesting a feature to be implemented or a defected to be resolved, he or she must simply post an issue, and has the opportunity to use a set of descriptive tags, including *help wanted* and *good first issue*, which are recommended issues for people looking for issues to fix. The process of communicating an issue to the VS Code community is given in a set of guidelines for contributors to follow. First of all, users are expected to search for the issue before posting and only post if the search is unsuccessful. Secondly, only one bug is permitted per report. Thirdly, the user must provide some key information: the version of VS Code along with any and all extensions installed, the steps necessary to reproduce the defect detected or witness or understand where the requested feature will occur and be used.

There are several guidelines regarding the actual code that users wish to push. Type and enum names use PascalCase, while function, method, property, and local variable names use camelCase. Moreover, it is expected that users pick whole words in names whenever it is possible to do so. Microsoft specifies that contributors should not export functions or types (unless it is to be shared across different components) or make anything global. Commenting is highly encouraged and done in JSDoc style.

Additionally, there are guidelines specifically to code style given. Loops and other conditional bodies should be enclosed by braces. Any time a comma, semicolon, or colon appears, there should be a whitespace that follows.

For a contributor to either push a newly implemented feature or resolve a reported defect, it is expected that before a pull request is made, he or she run the automated test script provided by VS Code. Also, Travis CI tests will be run automatically when a pull request is made.

To ensure good code style of submitted implementations and fixes, VS Code expects contributors to run TSlint on their code.

TASK DESCRIPTION

Triple Click Extension ID

In Visual Studio Code, when one would attempt to highlight and copy the ID of any given extension by triple clicking, a massive amount of extra text would be copied as well. This included menu and button names, and header titles. It was requested that this defect be fixed. Another user proposed one solution, but it was noted that this caused the same issue to appear elsewhere, just not with the extension ID.

We were able to resolve the defect by adding a user-select property to a parent class of the extension ID, which effectively allowed the user to triple-click to solely select the ID. We also needed to add the user-select property to the following tag and set it to none, so that the following tags did not have the same issue.

Disable Breakpoint Keybinding

While in debugging mode, a user is able to add and remove breakpoints, as well as select different breakpoints to enable and disable. In regular mode, however, a user only has the options to either disable or enable all breakpoints--it cannot be done individually.

Originally, the source code for VS Code separated the methods used by debugging mode and editor mode. That is, creating a breakpoint had two different methods: one for editor mode and one for debugging mode. The editor version of enabling and disabling breakpoints did allow for individual breakpoints to be disabled or enabled. Thus, instead of implementing a new version of the method for editing mode, we chose to rework the code to instead call the same method that was used by debugging mode, which had already been verified and tested, effectively adding the new feature.

Line-by-Line Selection in Terminal

When looking at output in the terminal, there are keybindings to select the previous or next command that was entered into it, but no keybinding to select the previous or next line (command or output). A feature request asked for these keybindings and the accompanying functionality to be added.

We were unable to fully implement this task, due to its complexity. We had assumed that they were tracking the lines in the terminal and it would be straightforward to increment or decrement the lines. However, they were apparently not tracking all of the lines in the terminal; they only tracked commands. Obtaining the line numbers of the lines between proved tricky. We added the keybindings for the commands but ran out of time to implement the actual functionality behind the keybindings.

SUBMITTED ARTIFACTS

Triple Click Extension ID

https://github.com/lwillia/vscode/tree/lwillia/extension-id-select leads to the branch where we pushed our changes.

https://github.com/Microsoft/vscode/pull/47921 leads to the pull request to merge this fix into their master branch. The pull request also contains evidence of the checks they ran on our code: Travis CI and another continuous integration check, AppVeyor.

Disable Breakpoint Keybinding

https://github.com/lwillia/vscode/tree/lwillia/debug-disable-breakpoint leads to the branch where we pushed our changes.

We did not submit this one, due to another VS Code contributor submitting a pull request before we finished.

Line-by-Line Selection in Terminal

https://github.com/lwillia/vscode/tree/lwillia/terminal-line-selection leads to the branch where we pushed our changes.

We did not fully implement this, so we did not submit it as a pull request. Moreover, another contributor submitted a pull request first.

QA STRATEGY

Testing

We used both Travis CI to perform very standard integration testing on the code we produced and successfully passed all cases. Beyond that, we performed a significant amount of static analysis to seek out any potential defects. All three of our tasks dealt with human input and thus, in order to fully test our implementations, we interacted with VS Code after our changes had been applied to demonstrate the behavior. Additionally, we ran the automated test suite included in the source code to verify that the code worked.

Styling

In an effort to hold our code readability to Microsoft's standards, we used TSLint as a style guide as well as VS Code's naming standards, space and tabbing standards, and style standards.

Reasons for choosing

We chose to perform these strategies for a few reasons, but mainly because they were very well spelled out and mandated by Microsoft as part of the guidelines for contributors. As a result of this, the readability will be very decent and completely match the styling standards that Microsoft desires.

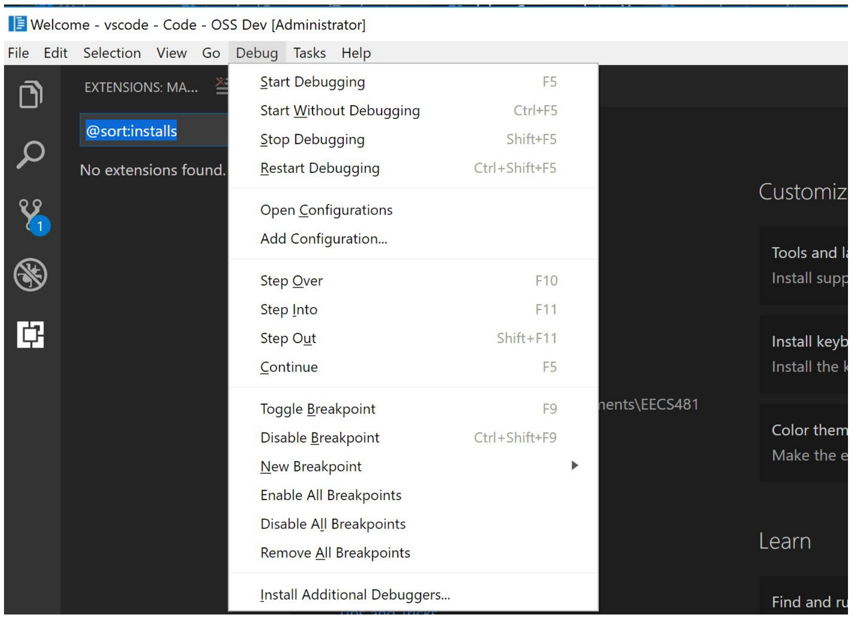
QA EVIDENCE

Triple Click Extension ID

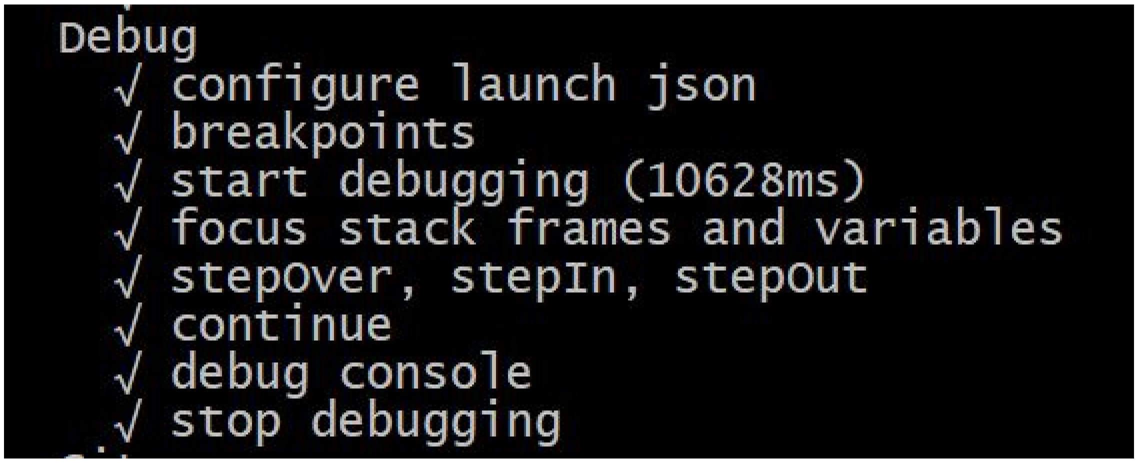
Successful Travis CI run:
https://travis-ci.org/Microsoft/vscode/builds/366944491?utm_source=github_status&utm_medium=notification

Disable Breakpoint Keybinding

The disable breakpoint keybinding is listed in the menu now:



We ran the included tests on our code and it successfully applied breakpoints, as seen in the screenshot below.



```
Debug
  √ configure launch json
  √ breakpoints
  √ start debugging (10628ms)
  √ focus stack frames and variables
  √ stepOver, stepIn, stepOut
  √ continue
  √ debug console
  √ stop debugging
```

Line-by-Line Selection in Terminal

As we did not complete our implementation, we also did not perform any quality assurance on it.

PLAN UPDATES

We deviated from our original plan, particularly with the line-by-line selection task. We started out with this task, but found it considerably more complex than we had anticipated. Once we realized this task was this complicated, we decided to implement the other tasks first. We came back to the line-by-line selection task toward the end of our work, but did not have a sufficiently long enough amount of time to complete it.

The triple click bug task was considerably easier to implement a fix for than we anticipated, but the debugging aspect of it took much longer than expected.

Our entire schedule was also shifted back several days, due to studying for exams and other projects.

Updated Schedule
Friday, 04/13 – Explore line-by-line section task and start implementing (4 hours)
Saturday, 04/14 – Identify bug and implement triple click bug task (4 hours)
Sunday, 04/15 – Triple click bug task QA (2 hours), explore breakpoint disabling task (2 hours), implemented breakpoint disabling task (2 hours) and perform QA on it (3 hours)
Monday, 04/16 – Continue implementing line-by-line selection task and report (8 hours)

This entire homework took roughly 25 hours for each of us.

EXPERIENCES AND RECOMMENDATIONS

Introduction

Contributing to an open source project was a very new experience and we learned a lot from it. Between the two of us, we have had experience working in industry on large software programs, but it was in a professional, structured environment. Having the opportunity to work on an open source project as large as Microsoft's Visual Studio Code, independently, with no actual deadlines--at least, none given by the actual company--was both a very foreign experience and enlightening one. Overall, we learned a lot from working on our contributions to Visual Studio Code, and had a wonderful opportunity to apply many of the concepts we had learned in EECS 481 to a real world scenario and fully demonstrate our software engineering prowesses.

Surprising Aspects

Over the course of reading the issues and implementing the resolutions, we actually came across multiple surprising aspects resulting from this process. One of the more prominent ones was the fact that some of the defect issues that are posted have very trivial solutions. We did learn, however, that this triviality does not imply a quick and easy resolution. As it turns out, a significant amount of time is spent resolving defects, some of which would have been very easy to fix if it were caught early in the game. It still took us a decent amount of time to locate and fix the defect, but the actual way to fix it was fairly short.

Difficulties with the Source Code

As we spent time examining and familiarizing ourselves with the source code, we ran into a few difficulties. One of the most frustrating of these was the lack of spacing between lines of code. While there was sufficient tabbing, this simply was not enough to compensate for the poor choices in spacing. We, on the other hand, made a decent attempt to space out anything that would require it to have as good of quality assurance as possible. Furthermore, there clearly was commenting throughout the code, but it was not what we would have considered sufficient. There was little commenting within functions. Additionally, none of the comments we observed included any sort of "why" statement, only "what" and not necessarily a good one. This made it particularly difficult to understand the code dealing with line selection in the terminal. The code regarding the terminal was a lot more complex than in the other two tasks. It was not very clear what data structures were holding what data. Moreover, several of the names were fairly ambiguous, despite the fact that, for the most part, they met the guidelines of using whole words when possible. One variable in particular was named **Boundary** and had a member variable, **Bottom**. We spent a decent amount of time attempting to learn whether this function call with this variable implied that we were looking at the last line of the terminal, or the last *visible* line of the terminal. Had the developers spent more time making the code readable, even if they were less certain if it was correct, we would have spent less time attempting to understand the code and had a much easier time correcting any mistakes and implementing the new feature.

Challenges Implementing and Submitting

As mentioned above, we were surprised by the lack of difficulties presented by actually submitting our changes to github. For the most part, it was straight-forward process and the steps to making the code ready for submission were very clear. The only real challenge we faced regarding submitting revolved around the fact that it was an open source project with several hundred contributors; we were not the only ones looking at this issue and another contributor submitted an implementation of the **Disabling Individual Breakpoints** feature before we were able to do so. Beyond that, however, it was fairly easy.

We did, however, run into a few issues actually implementing the code. Most of our problems were results of not understanding how the structure of the code worked. As was mentioned earlier, the readability of the code and the organization was a bit subpar. This made implementing the **Line Selection** feature significantly more challenging than the other two tasks. Despite the time spent familiarizing ourselves with the organization of the code, we were, in fact, not able to successfully complete this task.

Interacting with the Community

The community was active for all three tasks and each one had a decent thread of conversations and comments. There were several instances of members who, even without implementing the task themselves, were able to provide advice or guidance for what should be done, which was very helpful. Whenever we would make any sort of post, someone in the community was fairly quick at getting back to us with a response. Even when we made our pull request for the solution we had designed for the **Triple**

Click Selection task, a fellow contributor was very quick at letting us know that our solution “looks good to [him]” and thanking us before it was merged into the master branch.

Another concept we observed about the community was their development process. They worked in monthly iterations and planned out milestones. They triaged each issue and assigned it to a milestone, based on assigned priorities. They documented their development process on the VS Code wiki.

Changes to the Community Dynamics

While the community was, overall fairly well run, there were a few changes we would like to have seen while we were working. First of all, we really would have enjoyed seeing more commit messages that incorporated a “why” factor, as opposed to exclusively describing what was being done--there is readily available software that can describe what happened; we need the person to provide the “why.” This would have made trying to work off of another implementation that a different contributor was unable to finish a very reasonable task.

Recommendations

Holistically, the Visual Studio Code open source repository is very well-managed, but no project is without its flaws. While we were working on making contributions, we noticed that most of our problems revolved around the organization of the code. The lack of spacing and complex data structures presented a strong learning curve. The code is, however, meeting all of the *functional* requirements its been given thus far, but seems to be lacking a few properties that would (and should) have been considered quality requirements. As a result, the code works, but it is not very well designed for maintenance activities, particularly *comprehension*, *testing*, and *change*. As was stated previously, understanding the already existing code and attempting to add new features was difficult. It is for this reason that our biggest recommendation is to *refactor* at least some of the the code. Above all, extracting some of the larger classes would provide a significant amount of help in understanding the structure of the code.

ADVICE FOR FUTURE STUDENTS

You may let future students view our materials.

From Laurel:

> *Skimming the reading assignments is not sufficient for the reading quizzes, and they contain a lot of useful information for the exam.*

From Cody:

> *Understand that EECS 481 is about learning and will give you the skills to excel at*

interviews, so take your time to genuinely learn the material covered, speak with the staff about how you can use the information to improve your abilities personally, and enjoy your time!

EXTRA CREDIT

https://github.com/Microsoft/vscode/pull/47921

This link shows the pull request for our fix for the triple click on extension id that was merged into their master branch.