IDE Intelligent Tutorials (IDE-IT): Frontend

Motivation

Integrated Development Environments (IDEs) are software applications that provide tools and facilities for developers to write and test software. They can be immensely helpful to developers as they provide user interfaces and features for authoring, modifying, compiling, deploying, and debugging software. However, due to low accessibility and awareness of these tools and features, only a small number of these powerful IDE functionalities get used [1][3]. To clarify what is meant by accessibility, we mean that many useful configuration options are difficult to find and are several menus deep so they often remain unused or undiscovered by developers. Additionally, developers find that many tools in their IDEs are not trivial to configure - with confusing settings for each configuration option and too many words that make it challenging to understand what needs to be selected for the configuration option to work, and this prevents them from using the tool at all [2]. These studies present the issue of discoverability of existing IDE tools and shortcuts. Specifically discussing the Eclipse IDE, it has a vast plug-in ecosystem which offers rich rewards, but only for developers who know how to find these “gems”.

For the reasons previously described, we propose development of a tool which improves discoverability of existing tools, settings, and plugins for IDEs. Due to the team’s familiarity with Eclipse, the proposed tool is designed for that platform, though the approach is generic and can be extended to other IDEs. At a high level, IDE Intelligent Tutorials (IDE-IT) would be a tool that provides suggestions for hotkey shortcuts and Eclipse configurations. Composed of a backend plugin, which performs the evaluation of which hotkeys and configurations to suggest, and a frontend plugin, which provides the interface for suggesting these to the user, this proposal will focus on the frontend development. The frontend will provide an interface to easily toggle different configurations between enabled and disabled, such as recommended compiler warnings and various editors. The goals of our proposed project is to teach users about the features of the Eclipse IDE that they may not be aware of by presenting them in a user-friendly way which makes accessing and configuring these features a trivial task.

Related Work

The two closest existing plugins to accomplishing our goals are MouseFeed and Features Tainer. MouseFeed, a plugin for Eclipse, works by generating a popup notification any time the user clicks a button in the toolbar or a menu item, reminding them of the hotkey shortcut for that feature [4]. This works great if a user already knows that a feature exists, but falls short as a full solution as it requires the user to be aware that a feature exists in the first place. IntelliJ’s Features Trainer plugin is available to help learn shortcuts for the most used IDE actions [5]. It consists of 5 modules, providing sequences of tasks in which the user invokes the correct action shortcut to progress. This tool lacks in suggesting tools relevant to the user at the current time, and both MouseFeed and Features Trainer interrupt development flow. Additionally, most IDEs have some form of tips, usually tips of the day, that try and convey some of their features to the user. However, those tips are commonly irrelevant to the user, distracting, and interrupt workflow. Otherwise, web searches are currently the best way to discover useful tools and

plugins, again interrupting workflow. Thus, it's difficult to know a feature exists in an IDE for something that a developer would find incredibly useful, unless they stumble upon it in these various ways.

Emerson Murphy-Hill and collaborators used several existing command recommender algorithms to suggest new commands to developers based on their existing command usage history, as well as introduced several new algorithms [6]. While they did perform studies via making recommendations to real developers and asking for feedback, their only feedback was about the quality of recommendations. Users responded to questions about novelty and usefulness. Thus, this study was focused on algorithm quality rather than the user interface itself, as our team is interested in. While this study provided insight and results into successful algorithms for command recommendations, it fell short in analyzing distractibility, workflow interruption, and ease of acting upon the given suggestions. Our tool will be non-invasive, only showing relevant features in an integrated window of the IDE rather than having pop-up notifications. This will reduce distractibility and workflow interruption since the user can easily view the suggestions without requiring the user to act upon the notification.

Approach

The overall approach is to utilize plugin dependency, with our frontend plugin registered to listen to the backend plugin. The backend plugin will be responsible for collecting and monitoring user input through the IDE, running evaluation functions on this user input, and producing a list of suggested features (see backend team proposal). As the frontend team, our approach is to display these suggestions to the user in a window within their IDE. These suggestions will come in two forms: tool enable/disable and hotkey shortcut tips. The latter will be represented as a simple suggestion, likely with a light bulb icon, while the former will consist of a checkbox for enable/disabling (Figure 1). Both will have the ability to be closed and not shown again, through use of an exit button indicated with an "X".

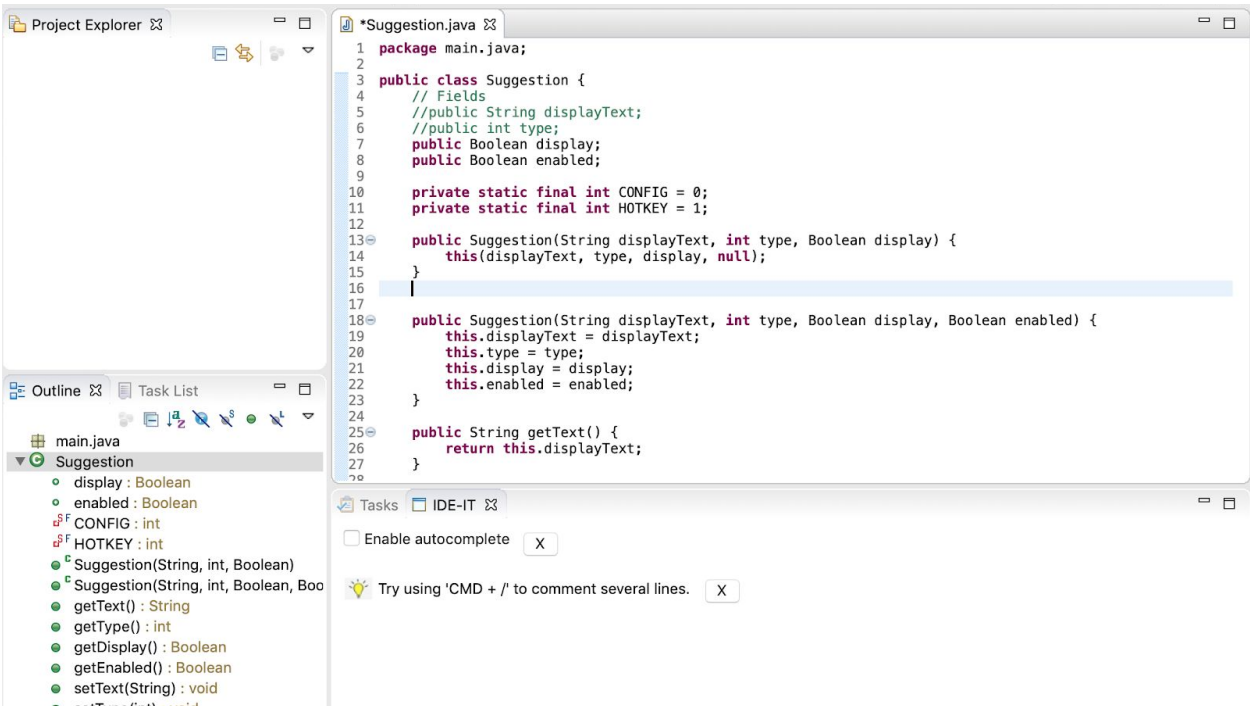


Figure 1. Current UI for IDE-IT suggestion window displayed in Eclipse workspace

The key difference between previous approaches to this problem and IDE-IT is that rather than teaching users more about features they already use, or relying on random daily “tips” to increase user awareness of features, IDE-IT will teach users about the existence of features that are actually relevant to the way they use Eclipse. IDE-IT frontend will rely on a backend service which tracks user action such as document changes, key presses, and mouse clicks, and report in real-time when it determines that features are not being utilized. Additionally, the notifications will be non-invasive, addressing the limitations of previous approaches such as distractibility and interrupting workflow. Through non-invasive notifications when users neglect to use relevant features, we provide a simple reminder that allows them to understand how they can make their work easier without interrupting it.

If successful, this plugin improving discoverability of other tools will help users to leverage features and inform users about faster ways to accomplish a task. Those who would benefit most from such a tool are the users who do not take advantage of many of the available features of Eclipse, especially for newcomers to Eclipse. Developers will benefit through a more enjoyable programming experience. Additionally, developers of Eclipse, its’ tools, and its’ configurations will care, as their tools will get more awareness and utilization.

Implementation

We have implemented a feature suggestion observer registered through the feature suggestion interface to be notified with features that the backend detects that the user may want to be aware of. Both frontend and backend have an identical resource, being a list of feature

identifiers: strings which uniquely identify each feature. Our frontend plugin receives a list of these strings via its FSObserver, which extends Feature Suggestion Observer from the backend (Figure 2). The following are descriptions of the classes shown in Figure 2:

- Feature Suggestion
    - Notifies its registered Feature Suggestion Observers with string identifiers
- FSObserver (extends Feature Suggestion Observer)
    - Abstract class that allows the frontend to be notified of the feature suggestions
    - Registered as an observer to the Feature Suggestion
    - Gets notified with string identifiers for suggested feature
- Controller
    - Creates a map of string identifiers to Suggestion objects
- Suggestion
    - Contains the text to display
    - Contains type (preference or hotkey)
    - Contains display boolean, indicating if the suggestion is currently displayed
    - Contains counter, increasing each time the suggestion is closed
- Main View
    - Creates corresponding ConfigDisplayComposites and HotkeyDisplayComposites
    - Populates the window with these displays, so the user can view them
- ConfigDisplayComposite
    - Creates an swt composite object, containing a checkbox, the suggestion text, and an exit button
- HokeyDisplayComposite
    - Creates an swt composite object containing a lightbulb icon, the suggestion text, and an exit button

After identifying the corresponding suggestion objects, these suggestions are then displayed to the user through our Main View class. Finally, the window is populated with the proper type of composite object. A composite object contains other widgets and is used to build more complex user interfaces. Based on the feature suggestion, either a hotkey display composite or a configuration display composite is shown in the window.

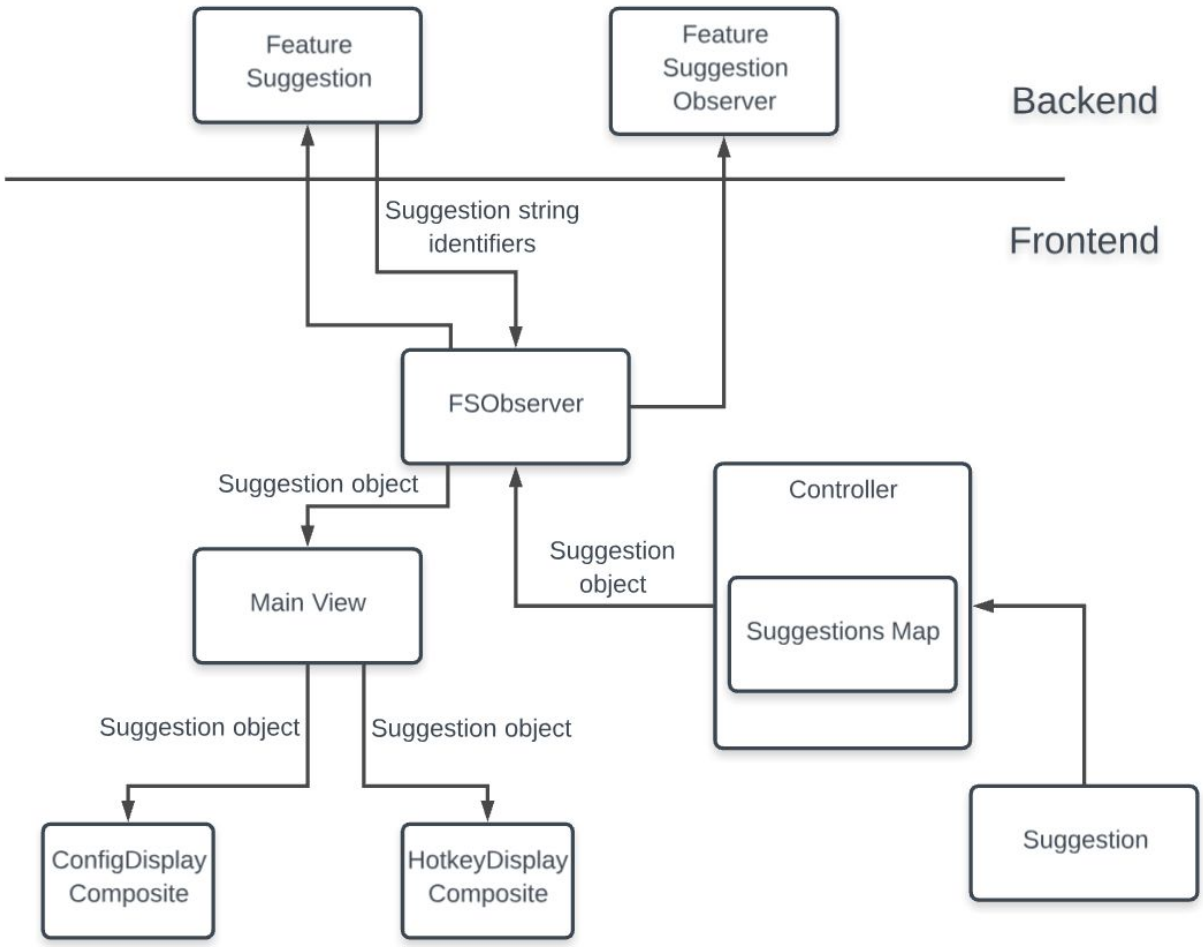


Figure 2: High-Level Architecture Diagram

Once a user has interacted with our suggestion window, whether via interaction with a checkbox or clicking the exit button, the frontend is also responsible for then handling the proper response. If a suggested tool has been enabled or disabled, that setting will then be updated accordingly. If the exit button was selected for a current tool, this likely indicated that the user does not want to see this suggestion again in the future. Thus, the display field in the suggestion object will be set to false, so that redundant or unhelpful suggestions are not displayed in the future. In this way, the suggestions will be less likely to annoy the user as current tool suggestion methods do.

For the first version of our plugin, we implemented a single suggestion: a suggestion to use the hotkey “CMD + /” for commenting out several lines of code. After successful implementation of this suggestion, we moved forward and implemented four more suggestions, chosen together by both the frontend and backend team. These next four suggestions are: 1) adding import statements using “CTRL + SHIFT + O”; 2) Removing unnecessary/unused import statements using “CTRL + SHIFT + O”; 3) Correcting indentation; and 4) Refactoring code base by renaming a variable throughout the entire project.

We are developing our plug-in in the Eclipse IDE, since we are building a plug-in for this environment. We will use Java to create the interface, make suggestion objects, use the data given to us from the back-end team, display the tips, and enable or disable configurations. We are also using the Standard Widget Toolkit in Eclipse for efficient and portable access to the user interface facilities. We will utilize the Eclipse user interface features to create elements like checkboxes, toggles, radio buttons, sliders, links, tooltips, notifications, and containers.

Evaluation

Throughout our development we have been testing our code for basic errors and exceptions, including null pointers, index out of bounds, illegal arguments, etc. We will ensure that when a user is using our plugin that no series of buttons clicks will result in it breaking. We will accomplish this by testing all of our methods and classes with their expected inputs but also with edge cases.

We have also done (and continue to do) testing through experimentation. This involves having ourselves and primarily other participants - ideally between 10-15 participants - use the plugin during development on their own projects. Through this we are testing that when someone chooses to enable/disable a feature from the suggestion box that it is correctly enabled/disabled. We understand that in the beginning phases of our development the features and hotkeys that we are suggesting might be trivial to more experienced developers so for our testing purposes, we will ask our participants to code as if they do not know how to block comment (i.e. type // in front of multiple lines), or not to use autocomplete shortcuts but instead type out many repeated words. This will cause those hotkeys and features to appear in the window so we can test their reaction to them as well as the functionality of selecting to enable/disable a feature.

The main way we are measuring the success of our product is if our experimenters reliably enable/disable the suggested features when a user selects one from the suggestion box. We will also want to make sure that when a user clicks the “X” - meaning they do not want to see the suggestion or tip - that we do not display it again. A failure is if the selected feature does not enable or disable. These successes and failures could be tracked for each feature and quantified.

One other important factor we need to test for is how the user responds to our interface. This includes the window placement, colors, icons, distractibility, clarity, font, etc. As we previously mentioned, developers don’t typically like many distractions or pop-ups and we want to blend this tool into their regular development process so we will be looking for feedback from participants as to how we can improve it visually. We are testing this by having people play around with our interface and then answer a survey about what they like and dislike. In this survey we are looking for rating from 1 to 5 (very unsatisfied, unsatisfied, neutral, satisfied, very satisfied) on how much they liked that factor. We also leave a section for general comments at

the end of the survey. Some sample questions our survey might include are (On a scale from 1 to 5): “How useful did you find this plugin?”, “How much did this plugin ease your coding experience on your project?”, “How likely are you to continue using this plugin?”, “How likely are you to recommend this plugin to other developers?”

Initial Results

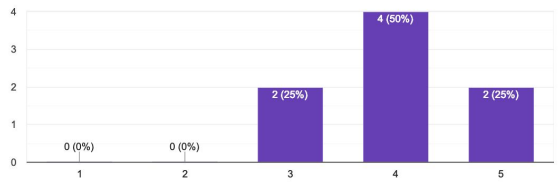
Through user testing we are able to test whether our product is accomplishing the goals that we originally set out for. After finding participants that fit our target group (developers who often use IDE’s - more specifically Eclipse), we asked them to write a couple lines of code in a workspace we have open for them. During the process, we asked them to perform several simple tasks and to pretend as if they are not aware of any hotkey shortcuts such as how to comment out multiple lines. Some of the tasks were writing print statements and accidently typing semicolons in the middle of statements. By having them code in such a way we are able to show them the full functionality and the many benefits of our plugin. Not only are able to see the hotkey suggestion appear in the window, but also check to enable config options that would make both of their tasks easier and quicker (content assist and smart semicolon).

After having (currently 8) participants test our product, we asked them to fill out a survey. This survey asked them a variety of questions surrounding their satisfaction with many features of our product, including config and hotkey suggestions, text size, icons, and overall design. The survey also includes questions related to development speed with the plugin and if the user thought it slowed them down or was in the least bit distracting or annoying. So far all of our results from the survey have been rather positive - with participants responding between 3-5 on questions with a 5 point scale, with 5 being very satisfied, 3 being neutral, and 1 being very unsatisfied, and response about development speed hindrances all being no. The survey ends with a comment section meant for any concerns or suggestions the participants have for us regarding the plugin. Most of the comments from our current participants were positive but some suggestions were to add more config suggestions and to fix some issues with the light bulb icon and the general display. Pictured below are some graphs representing the results from our survey. Note that all of these graphs are automatically generated and updated with every new survey submission by google forms and can be found on the owner of the surveys workspace at anytime.

IDE Intelligent Tutorials (IDE-IT): Frontend

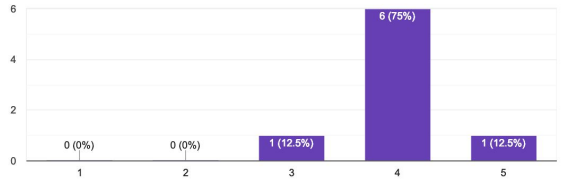
How satisfied were you with the config options that were suggested?

8 responses



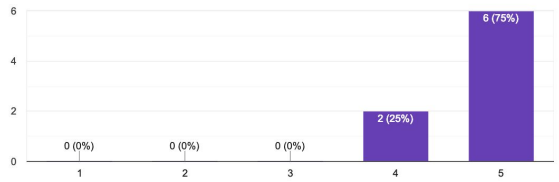
How satisfied were you with the design of the plugin?

8 responses



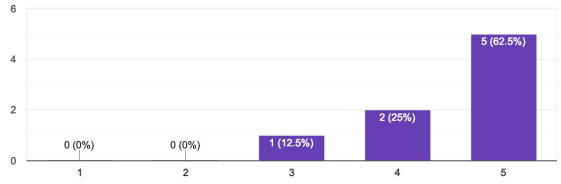
How satisfied were you with the hotkey tips that were suggested?

8 responses



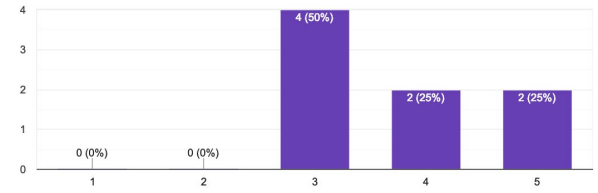
How satisfied were you with the text size of the suggestions?

8 responses



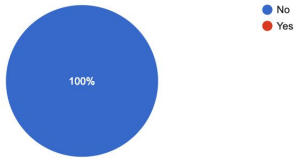
How satisfied were you with the icon design next to the hotkey suggestions?

8 responses



Do you feel as if this plugin slowed your development speed?

8 responses



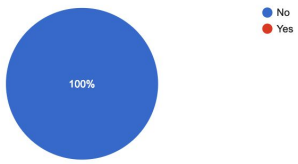
Please leave any comments, concerns, and/or suggestions for us so we can make this plugin better.

8 responses

More configs would be cool
It would be nice if the whole system.out.print showed up instead of system.out
I like the semicolon feature!
This is great! I already learned something! The icon could be better...
I often already have those config options enabled
Maybe clean up the formatting and make it look a little more polished
Auto complete paren if only one paren was shown. Smart Semicolon. What if I want to print a semicolon? Could have double tap semicolon fill the string field.
Put window on the side of the workspace to make the suggestions more noticeable.

Did you find this plugin distracting or annoying?

8 responses



If you are interested in more details about our user testing and/or survey you can visit our user manual (README.md) in our github repo to find instructions on how you could reproduce our current results - https://github.com/AlyssaRicketts/IDE-IT-Frontend. Additionally, those interested in reproducing and running the integration tests for this plugin should visit the SWT Bot Integration Tests branch of this repository - https://github.com/AlyssaRicketts/IDE-IT-Frontend/tree/swt-bot-integration-tests. The README.md in this branch contains information on how to run the integration tests.

Discussion

The initial results provided from our user testing provide insight into what our plugin currently does well, how it addresses the limitations of related work, and what current limitations still exist. The results indicate that our tool successfully addressed drawbacks of previous approaches in that all users did not find our plugin distracting or annoying, and also did not feel that the plugin slowed their development speed. Alternatively, as for limitations that still exist, users found that the icons, formatting, and design could be improved overall, as well as the configuration options that were suggested. In fact, one user mentioned that they often already have the suggested configuration options enabled. This is a very notable limitation of our current design; as the backend plugin does not currently support tracking document changes to suggest configuration/preference settings, we have three preference options that currently populate the IDE-IT plugin window upon opening it by default. These default suggestions fall short in providing a plugin which dynamically delivers relevant suggestions to a user. Thus, these default preference suggestions may be seen as annoying and irrelevant.

Another area for improvement in this tool lies in the number of times a suggestion appears after a user has closed it. Currently, hotkey suggestions appear up to three times after a user has closed it, if they close the suggestion but then continue to edit the document in a way that does not utilize the respective hotkey. This implementation was chosen over others, such as providing a second button stating “Do not show me this again”, as our goal was to implement an interface that is as simple and minimal as possible to avoid distraction and complication. However, suggesting the same hotkey tip three times could be considered annoying as well. We hope to address this limitation in our continued user testing.

Lessons Learned

We learned many lessons from planning, developing, testing, and documenting this project from start to finish. We found that allocating time for certain tasks was more difficult than anticipated since we would run into errors, lack of documentation on the topic, or other technical problems that slowed down the development process. Some of the tasks on the frontend side of the project that took longer than expected included setting up the configuration options, creating the build system, and working on the formatting of the window. We also learned that many of our ideas would not happen as we had planned; for example, we originally were planning on

receiving information from the backend to determine which configurations would display. Due to the amount of work the backend had to do to evaluate the document changes for just the hotkeys and the fact that many configurations are useful from the start to have enabled due to their generalized nature, we hardcoded in a few configurations that the user could enable and disable throughout their development process. This is also an initial stage, and we plan to have more configuration settings. With even more time, we would look into the backend providing these suggestions to display at a time relevant to the user rather than displaying them all upon opening our window. Through this experience, however, we learned that plans change but can always be revisited after the priority items have been completed.

We also found that creating task lists on a weekly basis and assigning these specific tasks to each person gave everyone on the team a clear goal to work towards. We started making these lists after the fifth week of the project, and they proved to be immensely helpful in increasing productivity and delegating work evenly amongst everyone in the group. This also provided a reference document, that was updated regularly after meetings with our manager or with the backend team, that each person could look at throughout the week.

Another important lesson we learned was fixing problems early on when they were smaller and more contained, rather than working on other tasks that may have been more prioritized at the time. Saving these tasks for a later time made the problem much harder to work on and also took much more time than it would have had we fixed the problem weeks before. One of the issues we had was using a starter “Hello World” plugin and having to refactor all of these references to be “IDE-IT-Frontend” later on. While this did not affect any functionality, we wanted to project to be cohesive and well maintained. Completing this task weeks later meant changing the build file for continuous integration and the build file for the Eclipse plugin itself.

Throughout this process, we communicated with the backend team every other week which was beneficial for making decisions about the interfaces and the details about each of the feature suggestions. We learned that having open communication between the teams allowed for the project to progress as a whole. Our meetings mostly consist of clarifying questions, giving updates, and discussing the integration between the plugins. Having this communication with the backend team was an integral aspect in the success of our plugin functionality.

Challenges and Risks

In developing IDE-IT frontend, the most difficult functionality to implement will be altering the IDE configurations. While many of the suggestions that can appear in the suggestion box are tips such as key shortcuts we will also be providing configuration options. If the users selects to enable or disable one of these options we need to successfully enable or disable that feature. The challenge here is that none of us have written a plugin before so we need to research how to access configuration menu options in code and make changes to the options as necessary.

Though there are no monetary costs in developing this plug-in, it will take time to build and test the tool to completion. We have created a plan for the next two months with the tasks needed to structure the front-end development. We anticipate to have a functioning interface in around six weeks, and we will use the following week to fix any issues that arise. In the case that we finish the basic implementation earlier than planned, we will add more functionality to the plug-in and refine our existing tool so that it is easy to use, intuitive, and helpful for the user during their development process.

Feedback

Some of the feedback we have received suggested editing out portions of our motivation section that the reader did not believe was relevant. For example, they mentioned that we begin by discussing accessibility and configurability but they do not believe this is what our tool addresses and suggest that we focus only on issues that our tool will solve. However, we disagree because the suggestions our tool provides that are paired with a checkbox are configuration settings. By checking (or unchecking) the box the user is able to easily enable (or disable) a configuration setting. We are targeting configuration settings that are often difficult to find but are extremely useful to developers - so by giving them a simple label with a checkbox we are increasing the accessibility of configuration settings.

Throughout our project development we have also received feedback that we have incorporated into our proposal and other documentation. Some examples are, splitting up our proposal into smaller more specific sections. We have already made changes inline with this suggestion as we feel it made our proposal more clear and stronger. We also changed our implementation diagram following some suggestions pertaining to it’s clarity and general correctness. In addition to these somewhat larger changes, we’ve also been editing each section every week to be more accurate in regards to our product. As we as a team change design ideas and product goals we make sure it is mirrored in our report.

Total time spent this week: 15 hours

Works Cited

1. Albusays, Khaled and Ludi, Stephanie. (2016). “Eliciting Programming Challenges Faced by Developers with Visual Impairments: Exploratory Study.” *IEEE/ACM Cooperative and Human Aspects of Software Engineering,* 82-85.

2. Johnson, Yoonki Song, Murphy-Hill, & Bowdidge. (2013). “Why don't software developers use static analysis tools to find bugs?” *Software Engineering (ICSE), 2013 35th International Conference on Software Engineering*, 672-681.

3. Kline, R., Seffah, A., Javahery, H., Donayee, M., & Rilling, J. (2002). Quantifying developer experiences via heuristic and psychometric evaluation. *Proceedings IEEE 2002 Syposia on Human Centric Computing Languages and Environments 2002,* 34-36.

4. *MouseFeed*. Eclipse Plugins, Bundles and Products - Eclipse Marketplace. https://marketplace.eclipse.org/content/mousefeed

5. Karashevich, Sergey. *IDE Features Trainer: a New Way to Learn Your IDE.* IntelliJ IDEA Blog - JetBrains. https://blog.jetbrains.com/idea/2016/12/ide-features-trainer/

6. Murphy-Hill, E., Jiresal, R., and Murphy, G. (2012). “Improving Software Developers’ Fluency by Recommending Development Environment Commands.” *SIGSOFT FSE.*

Appendix

A. Project Timeline

Week	Tasks
Week 3	- Project Proposal completed - Set up Git repo, mailing list, and communication methods - Determine each member's sub-team
Week 4	- Compile list of documentation and resources to use - Decide on basic design of user interface - Choose interface elements to include (input controls, navigational components, informational components, etc.) - Determine list of functionality we want to include - Determine what features to highlight and what their triggers will be
Week 5	- Implement a working prototype for what the interface will look like - Will not use backend functions yet - Test the prototype functionality
Week 6	- Continue implementation stated in week 5
Week 7	- Interface with backend to allow a functional prototype - Unit testing and set up continuous integration
Week 8	- User testing of functional prototype with surveys - Describe initial user experiment results - Write instructions to reproduce integration tests - Add different OS functionality for hotkeys
Week 9	- Window formatting and implementing user testing feedback - Continue to add additional tutorial functionality as needed - Complete rough drafts of final documentation - Further usability discussion and testing; fixing errors and bugs
Week 10	- Finalize everything - Refine specification - Prepare presentation materials