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 present the development of a tool with the goal to improve discoverability of existing tools, settings, and plugins for IDEs. Due to the team’s familiarity with Eclipse, the presented 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) is 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 report will focus on the frontend development. The frontend provides an interface to easily toggle different configurations between enabled and disabled, such as recommended compiler warnings and various editors. The goal of our 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 aims to be non-invasive, only showing relevant features in an integrated window of the IDE rather than having pop-up notifications. This reduces 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 is 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 come in two forms: tool enable/disable and hotkey shortcut tips. The latter is represented as a simple suggestion, with a light bulb icon, while the former consists of a checkbox for enable/disabling (Figure 1). Both have the ability to be closed and not shown again, through use of an exit button indicated with an "X".

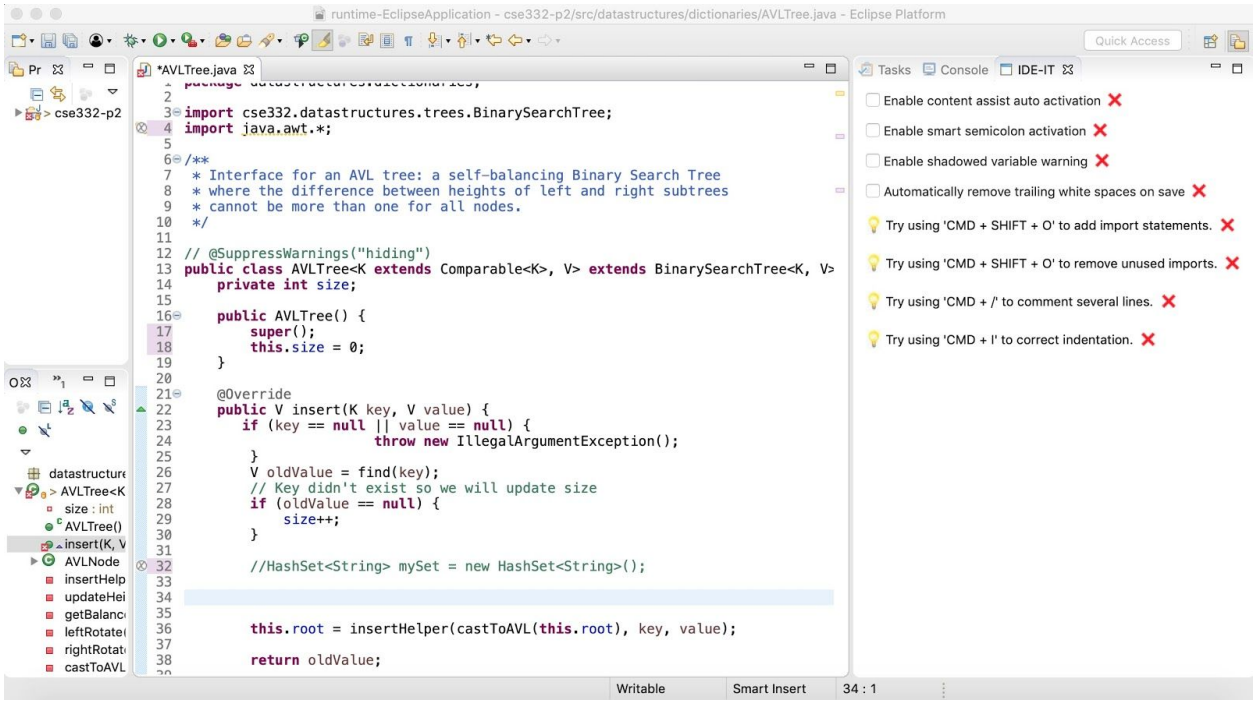


Figure 1. UI for IDE-IT suggestion window displayed in Eclipse workspace

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 teaches users about the existence of features that are actually relevant to the way they use Eclipse. IDE-IT frontend relies on a backend service which tracks user action such as document changes, key presses, and mouse clicks, and reports in real-time when it determines that features are not being utilized. Additionally, the notifications are 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.

This plugin that improves discoverability of other tools helps users to leverage features and inform them about faster ways to accomplish tasks. 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
- OSInfo
    - Extracts information about the operating system that Eclipse is currently running on

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 (Figure 1). 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.

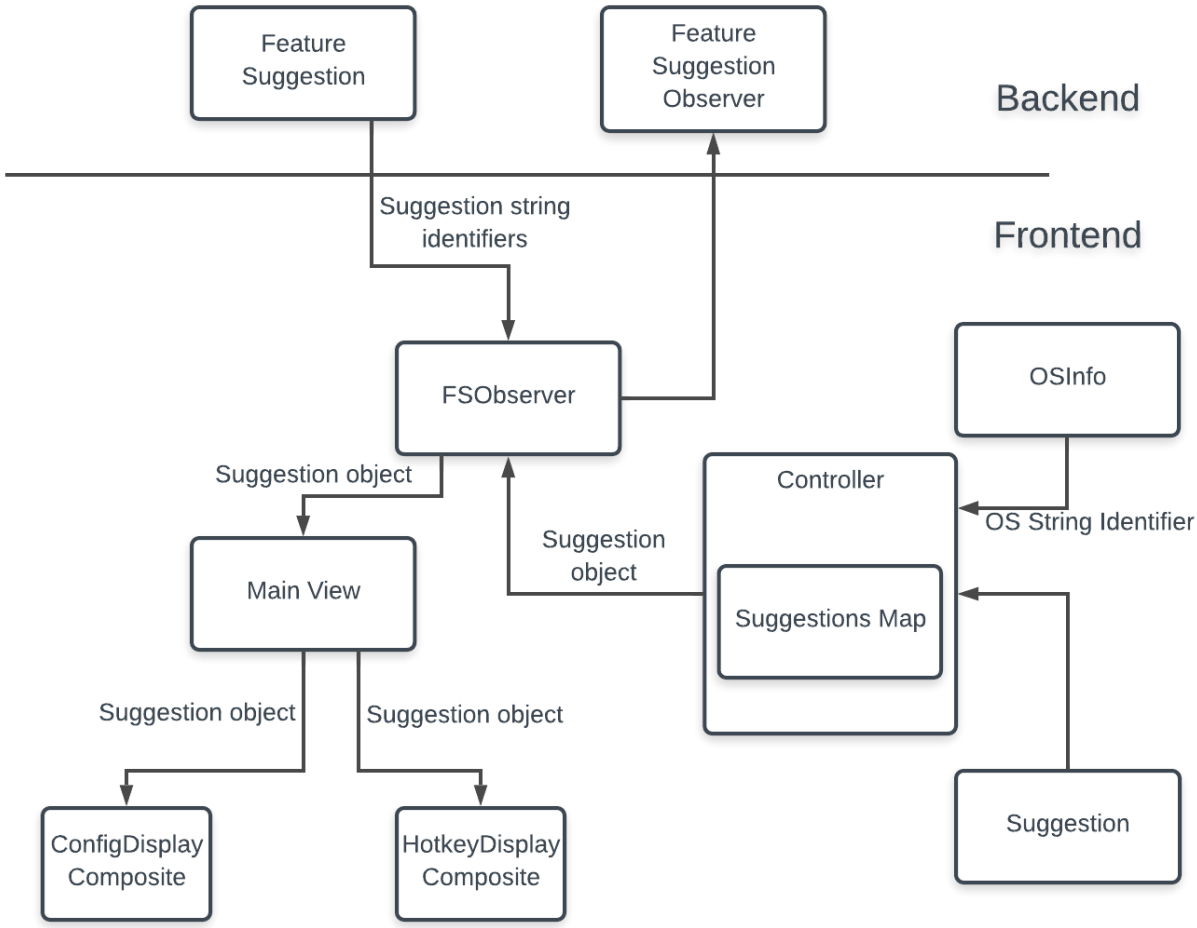


Figure 2: High-Level Architecture Diagram

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 is then 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 counter field in the suggestion object is incremented. Once a hotkey suggestion object has a count of 3, it will never be shown again, while configuration suggestions do not appear more than once. This way, redundant or unhelpful suggestions are not displayed in the future, so the suggestions will be less likely to annoy the user as current tool suggestion methods do.

For the current version of our plugin, we implemented nine suggestions, as follows:

1) Hotkey for block commenting
2) Hotkey for adding import statements
3) Hotkey for removing unused import statements
4) Hotkey for correcting indentation
5) Hotkey to automatically generate getters and setters

6) Preference to enable content assist autoactivation
7) Preference to enable smart semicolon
8) Preference to enable compiler shadowed variable warning
9) Preference to automatically remove trailing white spaces on save

We are developing our plug-in in the Eclipse IDE, as we are building a plug-in for this environment. We use Java to create the interface, make suggestion objects, utilize 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 also 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 tested our code for basic errors and exceptions, including null pointers, index out of bounds, illegal arguments, etc. We ensured that when a user is using our plugin that no series of buttons clicks will result in it breaking. We accomplished this by testing all of our methods and classes with their expected inputs but also with edge cases.

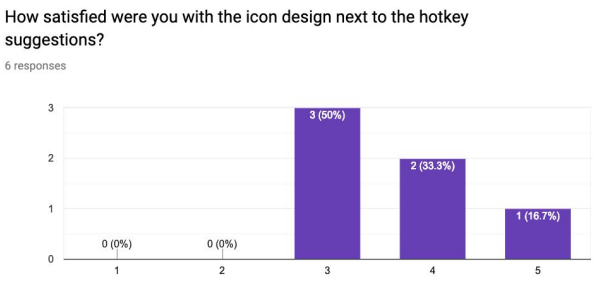
We also completed testing through experimentation. This involved having ourselves and primarily other participants use the plugin during development on their own projects. Through this we tested that when someone chooses to enable/disable a feature from the suggestion box, it is correctly enabled/disabled. 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 asked 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 caused those hotkeys and features to appear in the window so we we able to test their reaction to them as well as the functionality of selecting to enable/disable a feature.

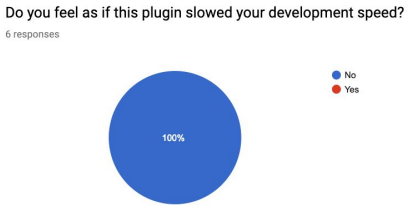
One other important factor we tested for is how the user responds to our interface. This includes the window placement, colors, icons, distractibility, clarity, font, etc. As we previously mentioned, developers don't typically like many distractions or pop-ups and we want to blend this tool into their regular development process so we looked for feedback from participants as to how we can improve it visually. We tested this by having people play around with our interface and then answer a survey about what they like and dislike. In this survey, we asked for ratings from 1 to 5 (very unsatisfied, unsatisfied, neutral, satisfied, very satisfied) on how much they liked that factor. We also left a section for general comments at the end of the survey. Some sample questions our survey included 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?"

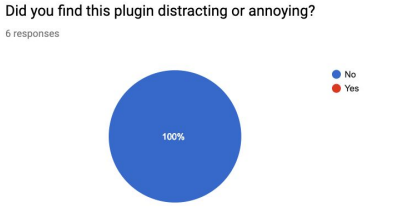
User Studies: Round One

Through user testing we were able to test whether our product accomplished 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 had 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 were able to show them the full functionality and the many benefits of our plugin. Not only were they 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 participants test our product, we asked them to fill out a survey: note that the round one results include 8 participants. This survey asked them a variety of questions surrounding their satisfaction with many features of our product, including configuration and hotkey suggestions, text size, icons, and overall design. The survey also included 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. All results from the survey were 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. All respondents indicated that it did not hinder development speed. 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 initial participants were positive but some suggestions were to add more configuration 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.



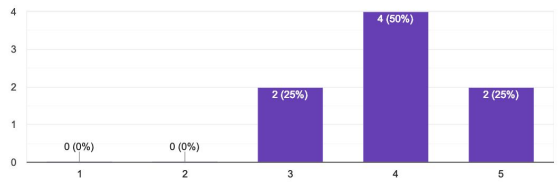




IDE Intelligent Tutorials (IDE-IT): Frontend

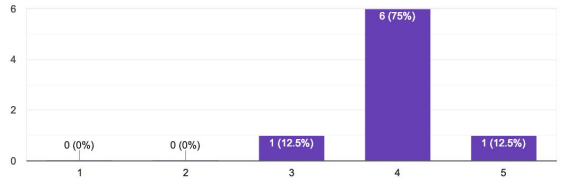
How satisfied were you with the config options that were suggested?

8 responses



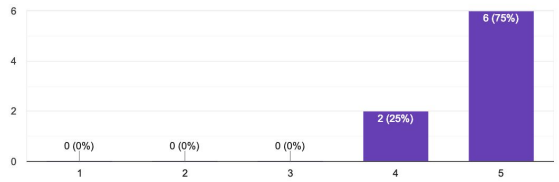
How satisfied were you with the design of the plugin?

8 responses



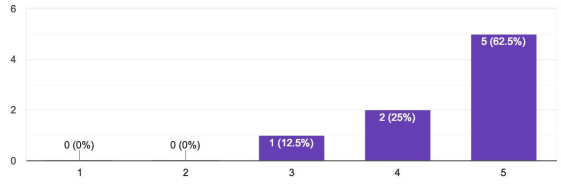
How satisfied were you with the hotkey tips that were suggested?

8 responses



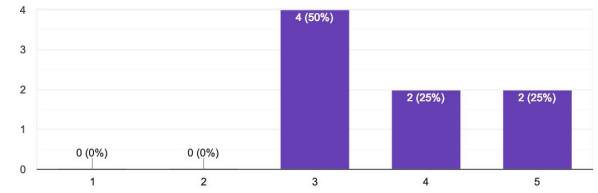
How satisfied were you with the text size of the suggestions?

8 responses



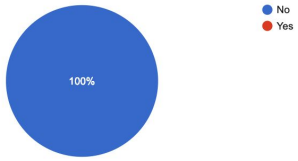
How satisfied were you with the icon design next to the hotkey suggestions?

8 responses



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

8 responses



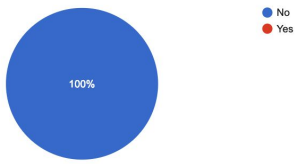
Please leave any comments, concerns, and/or suggestions for us so we can make this plugin better.

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



User Results: Round Two

After our initial user testing we made several changes to our product based on their feedback. Some changes were also made based on expected progress with our implementation. Thus, we performed a second round of user testing to analyze the improvements we made and continue to receive feedback. We performed this round of user testing the same way that we did with our first round, but our survey was slightly different just to reflect changes we made with the product. We still used a 1 to 5 scale (very unsatisfied, unsatisfied, neutral, satisfied, very satisfied) and are happy to report that all the responses were a 3 or above. We used our survey to ask our users what they thought about our choice to only display a hotkey tip 3 times before never showing it again; we understand that this was an arbitrary choice so we wanted to know if our users thought there was a better way to handle this. We have taken their responses into consideration and have decided to keep this implementation. The results from our survey are depicted on the following page.

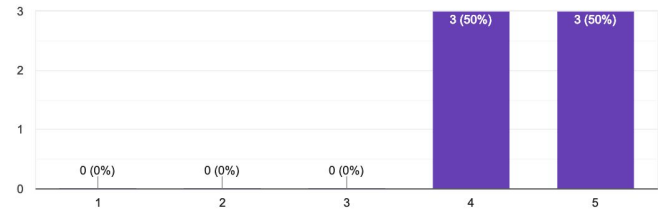
If you are interested in more details about our user testing and/or survey you can visit our user manual (README.md) in our github repo to find instructions on how you could reproduce our current results - https://github.com/AlyssaRicketts/IDE-IT-Frontend. Additionally, those interested in reproducing 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.

Both links above will take you to the Frontend developer repository, however the following link is what should be used if you are just looking to use our new tool: https://github.com/pujaram/Eclipse-IDE-IT. This repository contains both the Frontend and Backend code and all required dependencies to have our plugin working in your environment. Instructions on how to get it up and running can be found in the user manual (README.md) in the repository.

IDE Intelligent Tutorials (IDE-IT): Frontend

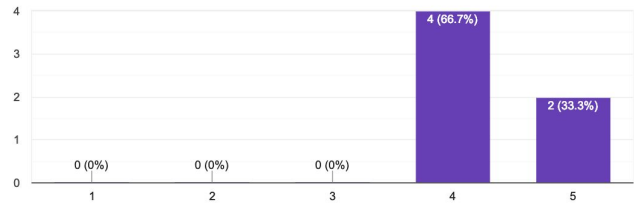
How satisfied were you with the config options that were suggested?

6 responses



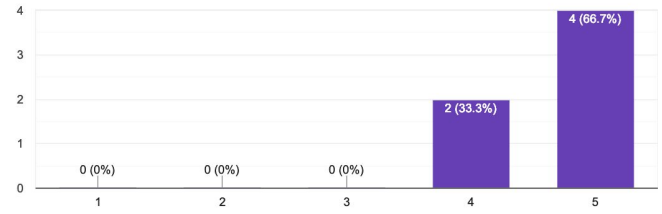
How satisfied were you with the overall design of the plugin?

6 responses



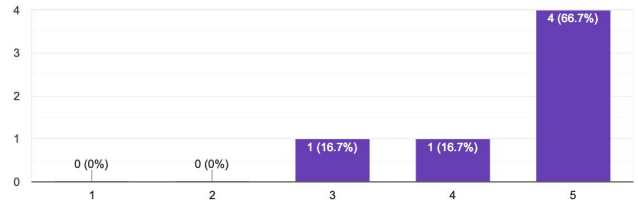
How satisfied were you with the hotkey tips that were suggested?

6 responses



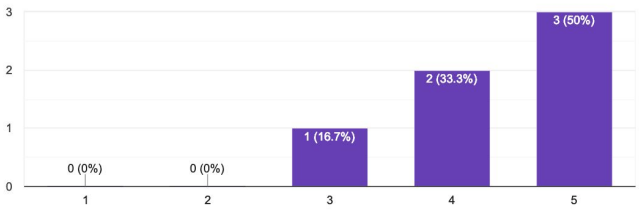
How satisfied were you with the icon design next to the hotkey suggestions?

6 responses



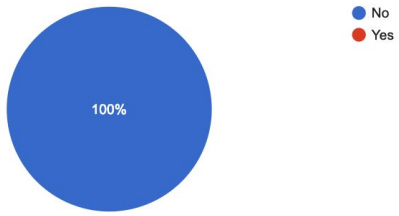
How satisfied were you with the style of the "X" button to remove suggestions?

6 responses



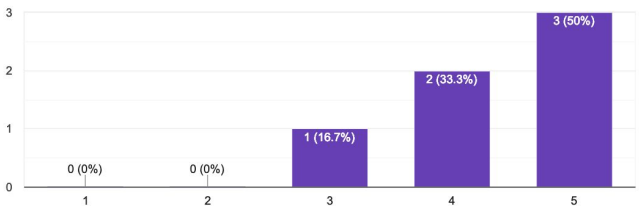
Do you feel as if this plugin slowed your development speed?

6 responses



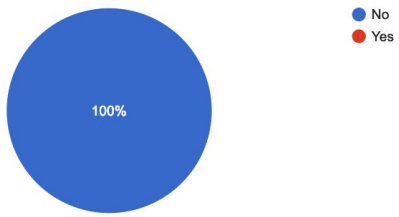
How satisfied were you with the text size in the window?

6 responses



Do you find this plugin distracting or annoying?

6 responses



How do you feel about a hotkey tip being suggested 3 times (if you "X" out of it and then continue to not use the hotkey) and then never again? Would you like it done another way?

6 responses

- no - thought it was good way to stop the nag. Only suggestion is to have a way to reset the "suppressed" items, if I decide maybe I suppressed things I don't want to suppress anymore.
- I think that's an awesome feature
- Maybe after you click the X once a popup could show up saying "Never show again"
- I think it's smart to keep showing it because it is an error that occurs and it offers a shortcut to help fix it.
- I enjoyed the friendly reminder, it only happens when I don't follow the suggestion. Can it be permanently removed as an option? Settings?
- It's helpful for some suggestions when I agree with them but makes for a large screen.

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

6 responses

- Think the plugin is useful for when I start to dev but as I get more familiar with the IDE, it may become less valuable. Very useful for beginners to the IDE.
- I think this is a great idea. This would be a very useful tool to have
- Some of the controls dont work but the design of it is easy to follow.
- Add a few more functionalities to it maybe
- Making the buttons more noticeable (mouse hover), I liked the lightbulbs because they immediately caught my attention. Very smooth interface with eclipse.
- It works well to improve use of Eclipse and take advantage of options I didn't know existed. Maybe grouping suggestion types could look better.

Discussion

The results from the two rounds of user testing provide insight into what our plugin 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.

Feedback
We have incorporated all feedback.

Total time spent this week: 25 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.*