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]. Additionally, developers find that many tools in their IDEs are not trivial to configure, 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, specifically Eclipse. IDE Intelligent Tutorials (IDE-IT) would be an Eclipse IDE plugin that provides suggestions for tools and shortcuts, and an interface to easily toggle tools between enabled and disabled. The goals of our proposed project is to teach users about the features of the Eclipse IDE that they may not be aware of. Our plugin will detect when users are not taking advantage of the many features it includes and inform them of how they could, thus providing relevant information at relevant times.

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. 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. 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, as well as interrupts 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, are randomly selected, and/or are more of an annoyance than a help. Otherwise, web searches are currently the best way to discover useful tools and plugins. Third party tutorials and tool suggestions are published in the form of long videos, cumbersome webpages, forums, and “Top 20 best features for…” articles. 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.

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 will continuously track 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. 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, as well as producing cleaner code the first time through by utilization of features. Additionally, developers of Eclipse and Eclipse tools and plugins will care, as their tools will get more awareness and utilization.

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 utilize a feature suggestion observer registered through the feature suggestion interface to be notified with features that the backend detects the user may want to be aware of. Both frontend and backend will have an identical resource, being a list of feature identifiers: strings which uniquely identify each feature. Our frontend plugin will receive a list of these strings via its subscription to listening to the feature suggestion backend. Upon retrieval of these string identifiers, the frontend will utilize an implemented map of string identifiers to “suggestion” objects. These suggestion objects will contain data such as the text to display to the user for this specific feature, whether this object is a tool or hotkey, what type of interface elements are included with this object (checkbox, link, toggle), and if this suggestion should be displayed in the future. After identifying the corresponding suggestion objects, these suggestions will then be displayed to the user. 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”.

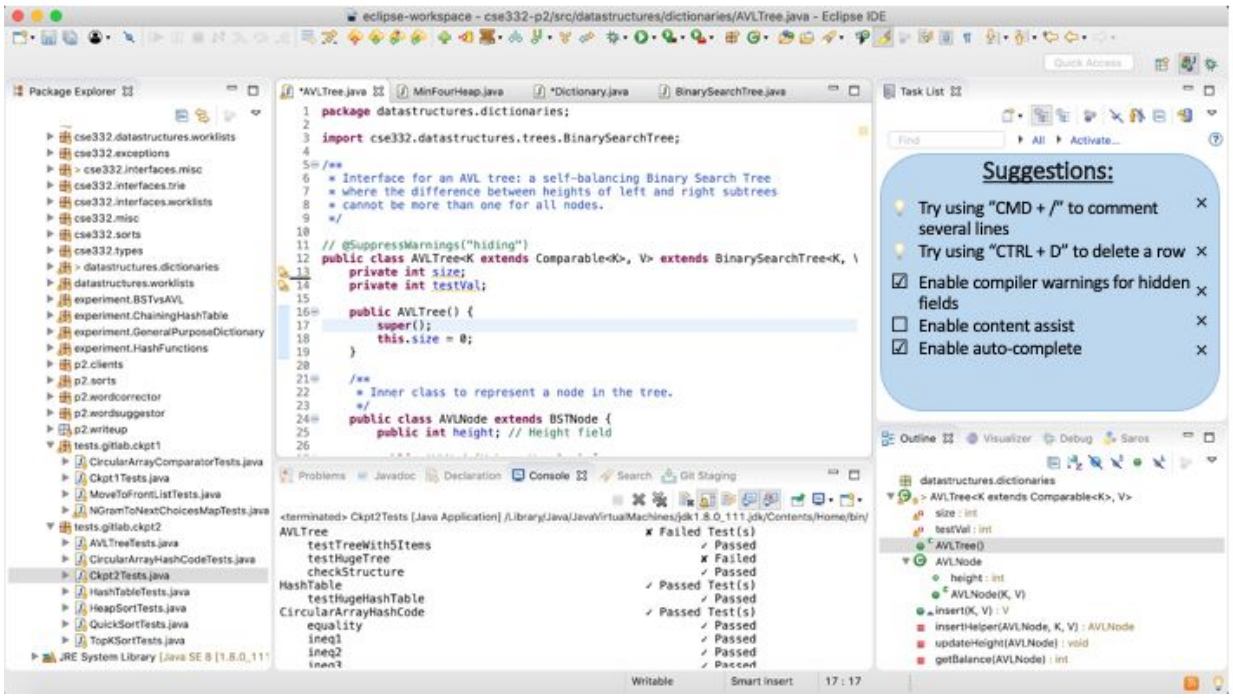

Figure 1. Mockup of suggestion box displayed in IDE

Once a user has interacted with our suggestion window, whether via interaction with a checkbox or clicking the exit button, the frontend will also be 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 will be implementing a single suggestion: a suggestion to use the hotkey “CMD + /” for commenting out several lines of code. After successful implementation of this suggestion, we will move forward to implement four more suggestions, chosen together by both the frontend and backend team. These next four suggestions will be: 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.

Outside of the standard unit and suite testing, we will do testing through experimentation. This will involve having ourselves and primarily other participants use the plugin during development on their own projects. Through this we will be testing that when someone chooses to enable/disable a feature from the suggestion box that it is correctly enabled/disabled. 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 may test this by having people play around with our interface and then answer a survey about what they like and dislike. In this survey we would be looking for rating from 1 to 10 on how much the liked that factor. We will also leave a section for general comments at the end of the survey.

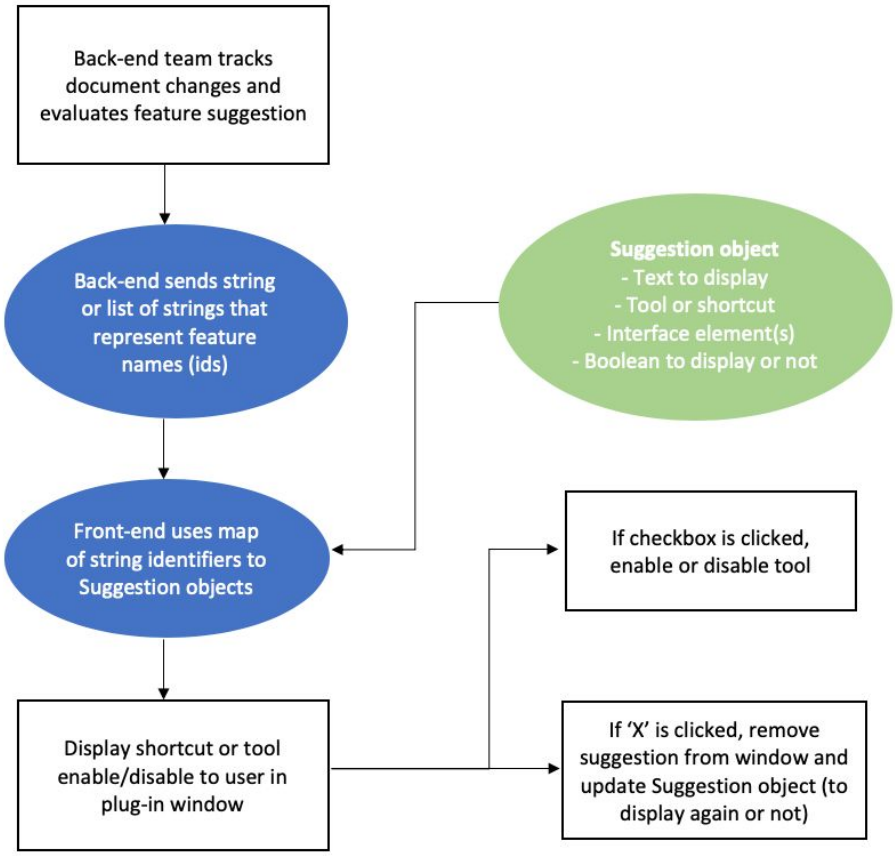

Figure 2: High-Level Architecture Diagram

The main way we will measure 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.

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.

On our front-end team, we have split up our project into three parts for each member on our team. Puja will be responsible for the user interface aspect of creating the window, checkboxes, and tips as well as the event listeners for all the interactive components of the plug-in. Alyssa will be in charge of creating the suggestion objects and creating the map of string identifiers to suggestion objects. Rachel will be responsible for the functionality of enabling or disabling certain configurations after the user checks one of those boxes.

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.

Proposed 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
Week 8	- Fix any issues reported from previous week - Include additional tutorial functions - Go through thorough review process with the team to discuss current usability - Determine what is feasible to fix and what is not in time remaining
Week 9	- Polish: include updates from previous week's discussion - Continue to add additional tutorial functionality as needed - Complete rough drafts of final documentation - Should have an almost fully functional plugin - Further usability discussion and testing
Week 10	- Finalize everything - Refine specification - Prepare presentation materials

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.