Discoverability of Existing IDE Tools

By Alyssa Ricketts and Rachel Zigman

Integrated Development Environments (IDEs) are software applications that provide tools and facilities for developers to write and test software. The cost of an IDE is not insignificant, and unfortunately there is evidence that this investment does not always pay off. Results of surveys suggest that developers often use only a small number of IDE functionalities out of the total set available [4]. Kline also reported that other issues in percent use of existing tools include low affordance of IDE menus, icon, toolbars, and graphical representation of program structure [4]. Another study found that developers may not use tools and features built into their IDEs because they are unaware of the variety of tools offered within the IDEs, find the tools to be too complex, or the tools are not easily accessible [1]. Likewise, developers find that many tools in their IDEs are not trivial to configure, and this prevents them from using the tool at all [2]. In this case study, developers reported that sometimes it is difficult just to get to the menu where the options for configuring a particular feature are, and developers shared stories where they could not figure out how to customize a tool and ended up having to search the web to find out where the tool's preferences were. The common issue that all of these studies discuss is the discoverability of IDE tools and plugins that already exist, and that if a developer does not know what their current issue or bug is, then they cannot know what to search, what features to turn on, and/or what plugins to download. 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". Thus, we propose development of a tool which improves discoverability of existing tools, settings, and plugins for IDEs, specifically Eclipse.

While the internet has many articles and forums for what tools are useful in what scenarios, it is challenging to realize what your current problem is in the first place, let alone what tools/settings in your IDE would help address these problems. Currently, web searches for these forums are the best way to discover useful tools and plugins. One tool in Eclipse somewhat similar to this proposal is the "content assist" which is mostly for suggestions as in auto-completion and variable names, not suggesting tools or setting changes. A different user interface toolkit, Path Projection, developed by Khoo et al. uses program visualizations to help developers walk through the error reports produced by static analysis tools [3]. Otherwise, there are many tutorials, step-by-step guides, and videos for IDEs online, as well as forums which answer questions developers might have about what tool is best for their given scenario. However, none of these options address the issue if a developer doesn't know the issue at hand, they cannot know what keywords to search, settings to turn on, or plugins to download.

A tool that improves the discoverability of existing tools and plugins would be valuable in reducing time writing code, debugging, and testing. As static analysis tools, debugging tools, etc. would be more readily available and accessible, developers would be more aware of their existence and thus more likely to use them. It would not only save time, but improve the percentage use of the IDE, thus making money spent on it more worthwhile. Developers would benefit by the tool, and it would improve speed and quality when writing code, as the full potential for the IDE could be in use.

Our high-level approach for a solution to this issue is an intelligent user interface that assists developers by providing suggestions for helpful tools and plugins. This assistance could ideally recognize the developer's current use and produce suggestions for tools as well as immediate links to enable/disable that tool, i.e. it doesn't require the developer to open a new

window or navigate through the IDE menu tabs. For example, if you are debugging a lot and investigating a specific variable while debugging, the assistance tool could see that you have certain warnings/errors ignored or tools turned off which would be helpful towards that specific variable, and suggest enabling them. Some key distinctions to separate our approach from other similar approaches are that we will use eclipse recommended settings, such as recommended compiler warnings [5] and top 10 customization of Eclipse settings [6] to tailor the suggestion tool, prioritizing these recommended settings first in the tool's suggestions. Limitations to our approach include consideration that developers do not like to be slowed down or interrupted, specifically they do not like opening another perspective or window to use a tool [2]. Thus, this tool would need to seamlessly integrate into development, providing suggestions but requiring as little navigation as possible to be done by the user.

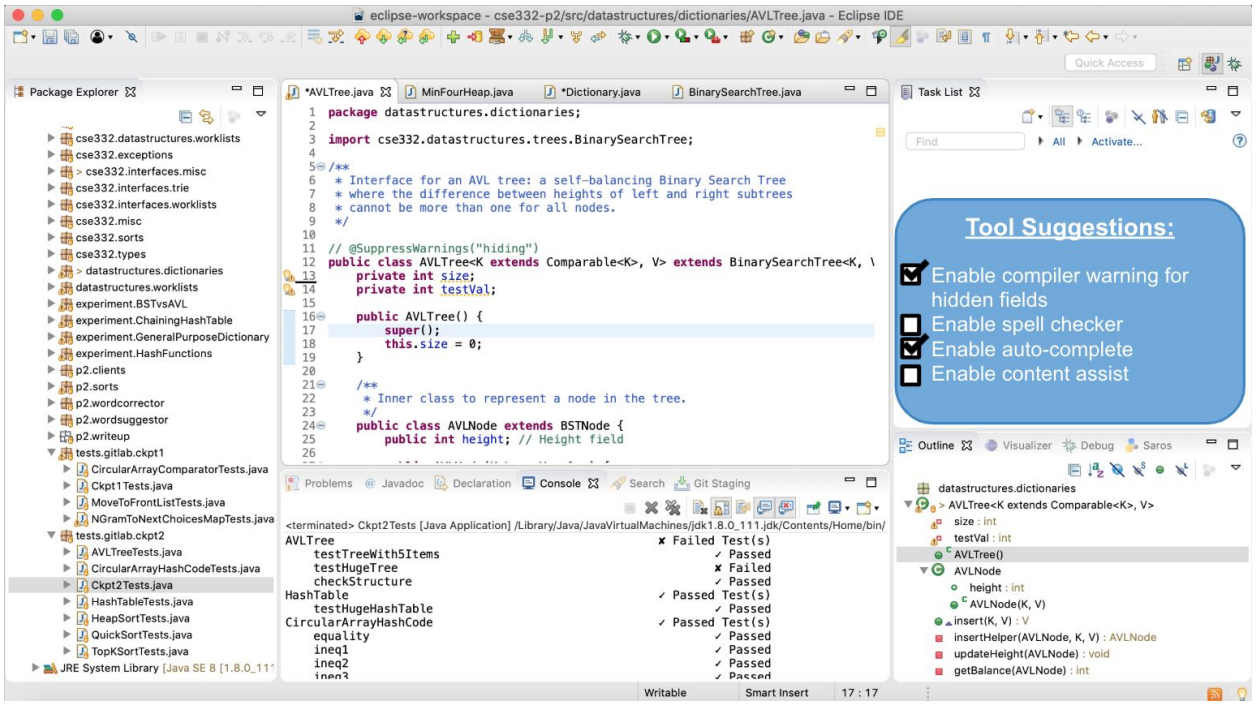

Figure 1: Mockup of proposed assistance for suggesting tool use in Eclipse IDE.

One of the most serious challenges we see in developing this product would be the accuracy of interpreting what the developer is currently trying to do, struggling with, or lacking, and identifying the best tool that could help them in their current scenario. Recognizing specific patterns in use and making valuable suggestions will be difficult. We can minimize this risk by providing a rating scheme for developers to rate the suggestions. This way, feedback can be given on if suggestions are good or bad so the tool could learn more about the user's specific project and then make better future suggestions.

Estimate of time spent on assignment: 6 hours

Discoverability of Existing IDE Tools
By Alyssa Ricketts and Rachel Zigman

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. Khoo, Y., Foster, J., Hicks, M., & Sazawal, V. (2008). “Path projection for user-centered static analysis tools.” *Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on program analysis for software tools and engineering,* 57-63.

4. 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.

5. Oberhuber, Martin. “Recommended Compiler Warnings.” *The Eclipse Foundation.*

6. Styger, Erich. (2013). “Top 10 Customization of Eclipse Settings.” *DZone.*