Problem-Solving Applications in Developer Environments

Nicholas Nelson
Electrical Engineering & Computer Science
Oregon State University
nelsonni@oregonstate.edu

Abstract

Programming is inherently a problem solving exercise: A programmer has to create an understanding of the situation, externalize and contextualize thoughts and ideas, develop strategies on how to proceed with the task, enact changes according to the most appropriate strategy, and reflect to learn from each problem. Therefore, programming is clearly more than just code input, testing, and maintenance. However, modern development environments largely focus on the "writing code" parts of programming. To support all aspects of problem solving in programming, we propose a new Integrated Development Environment (IDE) which uses a dynamic, expressive, and human-centric cards and canvas paradigm.

1. Introduction

Modern Integrated Development Environments (IDEs), such as Eclipse, IntelliJ, and Visual Studio, rely upon a fairly uniform interface of panes and windows to contain code and support different development lifecycle tasks. However, recent shifts toward distributed, service-oriented, and highly parallel software development have seen IDEs struggle to coup with the needs of developers. These developers have increasingly demanded tools that provide realtime feedback which integrates with both collaborators and customers that are increasingly embedded within the development model.

The pressure to add features that accommodate these interconnected development models has forced several IDE developers to rethink the core architectural designs of their environments. We believe that an entire reimagining is necessary in order to develop IDEs into general-purpose Problem Solving Environments (PSE).

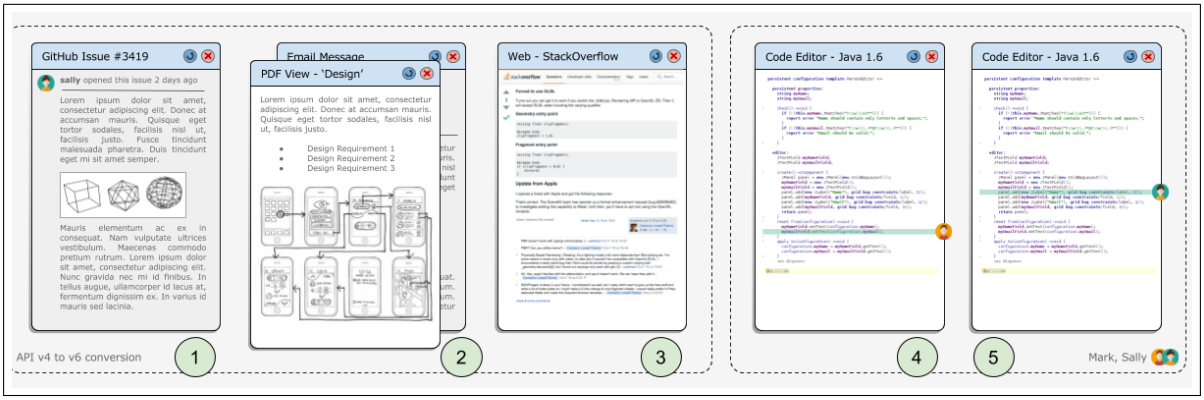
2. Research Approach

We first surveyed the literature from the perspective of programming as problem solving, and found several activities that developers employ when programming. These activities which can be partitioned into six categories (*Activities*), with specific actions that represent in more detail how the high-level activities manifest themselves in practice (*Actions*). Clearly, not every task involves all of these problem solving actions, and there is no linearity to the order in which they are employed. Sometimes an action may not even be observable when it takes place solely in a programmer's head. To support programming as problem solving, we proposed a new kind of IDE; a card-based IDE.

Our preliminary work will be utilized as the foundation for our IDE, which will iteratively focus on providing support for problem solving activities as development matures. We are iteratively designing this cards-based IDE, with the final goal of comprehensively supporting problem solving in programming, While doing this we will build and extend existing work, namely Code Bubbles (Bragdon et al., 2010), Variolite (Kery, Horvath, & Myers, 2017), PatchWorks (Henley & Fleming, 2014).

The IDE will be composed of an open canvas containing cards, which come in a variety of different types (code, text, sketches, web resources, etc.) and can be stacked and grouped according to the problem solving needs of developers (see Figure 1). Each card include a top bar containing a title and controls for going into fullscreen mode or closing a card. Within each card is multiple card faces representing different aspects of the same content (metadata regarding interactions with that particular card, static analyzer output for a piece of code, or links to references made throughout a GitHub project in regard to the issue contained within that card, different versions of a file saved within version control), which can be accessed using a swipe motion either via touchscreen or mouse.

Figure 1 – Cards-based User Interface of a Problem-Solving IDE



For example, as a user, Sally has a GitHub issue card open in order to gain an understanding of the problem that she is preparing to resolve (Figure 1-1). Swiping through the related faces of the card, she finds links to both a design document and an email that are related to this issue; she opens them into separate cards and groups them into a stack (Figure 1-2). She needs to alter the current design, so she makes some sketches directly onto the design card. While exploring potential solutions, Sally finds a StackOverflow posting that contains sample code that would be helpful (Figure 1-3). She moves that postings into a card and groups it with the GitHub issue and the previously stacked design and email cards, and adds a quick annotation to provide contextual relevance to other developers and herself.

Once she has developed a strategy, she opens the relevant code file into a card and begins enacting the changes necessary to resolve this issue (Figure 1-4). As her solution takes shape, Sally realizes that her changes are large enough to require a code review. She selects the two code cards that contain her current solution and shares them with Mark (Figure 1-5). During their review session, Mark and Sally both make changes to the code and are able to simultaneously keep track of each others progress and focus on their own code.

In this new environment, new cards are generated by opening files, selecting and extracting content from other cards, or via card sharing with collaborators. This sharing functionality provides a permissions model that allows developers to share specific cards (or groupings of cards) either in a read-only mode which retains control of the content for the owner, or a synchronized editing mode that allows simultaneous updates to occur from all collaborators.

Our goal with this IDE is to explore a new paradigm of interactions. The types of interactions that best facilitate problem solving is an open problem. Therefore, we plan to conduct several user studies regarding the different potential interactions made possible by a cards-based IDE. We hope to develop both a platform for future research, and a development environment that addresses the problem solving needs of real-world developers.

3. References

Bragdon, A., Zeleznik, R., Reiss, S. P., Karumuri, S., Cheung, W., et al. (2010). Code Bubbles: A working set-based interface for code understanding and maintenance. In *CHI* (pp. 2503–2512).

Henley, A. Z., & Fleming, S. D. (2014). The Patchworks Code Editor: toward faster navigation with less code arranging and fewer navigation mistakes. In *CHI* (pp. 2511–2520).

Kery, M. B., Horvath, A., & Myers, B. A. (2017). Variolite: Supporting exploratory programming by data scientists. In *Chi* (pp. 1265–1276).