Recommending More Efficient Workflows to Software Developers

Dylan Bates
Coker College
dylan.bates@coker.edu

ABSTRACT

Existing recommendation systems can help developers improve their software development abilities by recommending new programming tools, such as a refactoring tool or a program navigation tool. However, simply recommending tools in isolation may not, in and of itself, allow developers to successfully complete their tasks. In this paper, I introduce a new recommendation system that recommends workflows, or sequences of tools, to developers. By learning more efficient workflows, the system could make software developers more efficient.

1. RESEARCH PROBLEM AND MOTIVATION

Software developers use tools in order to make programming easier, by reducing development time and improving software quality. However, software developers only use a small fraction of the total number of tools available to them [4]. Several programs combat this by recommending tools to the user. Some of these programs work by simply recommending the most popular tools in an integrated development environment that the user does not make use of [2]. Others work by comparing a user's tool frequencies to the entire user population [3]. Exposure to more tools encourages the developer to use these tools to reduce development time and ultimately, increase software quality.

The sequential use of several tools characterize a software developer's workflow, which Linton and colleagues define as "one or more plans for attaining a goal," which "ultimately decompose to a sequence of actions" [2]. In this paper, I interpret this as a series of tools used one after another, in order to accomplish a specific, unique, and efficient task. I propose building on prior work that recommends tools, by instead recommending more efficient workflows.

Consider the following example of an inefficient workflow. Suppose Evelyn is working in the Eclipse development environment,[1] repeatedly using the `Find References` command on several methods in a call chain. In this example, the task she is trying to accomplish is walking up a call hierarchy. This is an example of an inefficient workflow, because she could accomplish the same task by using the `Call Hierarchy` tool, which takes fewer steps than using `Find References` repeatedly. Other efficient workflows use several tools, such as `Copy/Paste` to duplicate and move text and `Organize Imports/Format/Save All` to clean up code.

The insight of this paper is that I can use some of the same ideas to recommend workflows as were previously used to recommend tools.

My paper makes three contributions: it introduces a system that recognizes the most common workflows comprised of n tools; it demonstrates how n-tool workflows (called *n-flows*) can then be used to recommend a workflow to a software developer; and it explains how a recommendation system could implement this technique of recommending workflows, potentially increasing developers' efficiency and productivity.

2. BACKGROUND AND RELATED WORK

The previous work in this field recommends single tools (which I define as 1-flows) to a user. Existing systems do this via one of several algorithms, such as content-based filtering [3, 4], collaborative filtering [3, 4], and most popular [2].

The main difference between each of these methods is the way they generate the items to recommend. These methods have already been used by Matejka and colleagues [3], Murphy-Hill and colleagues [4], and Linton and colleagues [2] to recommend tools. One could feasibly extend their methods to recommend workflows to developers.

These methods are focused on recommending single tools that will potentially increase the productivity of a developer. The problem with existing implementations is that given out of context, some tools do not constitute a complete task, and are therefore difficult for a user to implement effectively. This problem was addressed by Viriyakattiyaporn and Murphy [5] when they created a system called *Spyglass*, which attempts to recognize inefficient navigational workflows and suggests tools to aid program navigation as a developer works. Unfortunately, Spyglass is limited in that can only recommend single navigational tools, instead of potential workflows that users could implement.

[1] `http://www.eclipse.org`

3. APPROACH AND UNIQUENESS

I went about the task of identifying the most common workflows by analyzing about 23 million time-stamped tool uses from 4308 Eclipse users, collected from the Eclipse Usage Data Collector.[2] A total of 700 unique tools were used. I attempted three ways to discover the most common n-flows, each of which has its own advantages and disadvantages.

These methods consisted of a Top-K Sequential pattern mining algorithm called TKS [1], sorting an n-dimensional matrix, and using a `Map` to map n-flows to the number of uses. Each implementation worked well for different sized datasets and numbers of dimensions.

Through the use of all three algorithms, I was able to determine the top n-flows for $n < 5$. I was then able to use Linton and colleagues' *most popular* algorithm [2] to recommend workflows to any given user in the set.

By looking at n-flows instead of single tools, this system is able to recommend entire workflows to a user that they may not otherwise discover on their own. For example, if Evelyn is new to Eclipse, and using an existing recommendation system to learn new tools, it will recommend a single tool to her (for example, `Copy`). Without the proper documentation for the tool, Evelyn may be confused. Certain tools, when used make no observable changes to the screen; she will find that nothing apparent happens. Had Evelyn been recommended a workflow containing the tool (in this case `Copy/Paste`), it may be more clear what the workflow accomplishes. In this case, the context for use is then clear. This method can eliminate the *discoverability barrier* [4] presented by Murphy-Hill and colleagues, as developers will learn new tools as well as more efficient workflows.

When cleaning the data, I had to prune the dataset to remove all repeated uses of tools, such as deleting an entire line one character at a time, or saving seven times before doing anything else. Before doing this, the top five workflows for $n < 5$ all consisted of repeated uses of a single tool; for example `Delete/Delete/Delete` or `Save/Save/Save/Save`. These were removed from the results.

4. RESULTS AND CONTRIBUTION

My research suggests that while there are several common workflows that Eclipse users implement in order to accomplish a task, there are many others that do not appear to serve a specific purpose. I collected the top 100 n-flows for $n < 5$,[3] and was able to generate 209 workflows and individual tools to recommend to a user in the dataset. I found that many of the n-flows I discovered did not meet my definition of a workflow, as they did not appear to accomplish a specific, unique, or efficient task. Due to this, most should not be recommended in a system that recommends more efficient workflows.

While 2-flows occurred most frequently, they are also the most trivial. For example, the most common 2-flow was `Copy/Paste`, which $>$ 99% of all users in the dataset used. The most common 3-flow was `Paste/Copy/Paste`. The most common 4-flow was `Copy/Paste/Copy/Paste`. Each of these n-flows is a subset of an $(n + 1)$-flow, which was found to occur throughout the n-flows discovered. Interestingly, the top eight n-flows for $n < 5$ consisted of various permutations of the following tools: `Copy`, `Cut`, `Delete`, `Paste`, and `Save`. In fact, most of the n-flows I discovered consisted of these five tools, as well as simple text editing tools, such as `Go To Line Start`, `Select To Line End`, `Go To Previous Word`, and `Select Next Word`.

[2] `http://www.eclipse.org/org/usagedata/index.php`
[3] Available at `http://goo.gl/nRKMv3`

4.1 Limitations

The major limitation that I faced was the noise found in mining common workflows, which makes recommending more efficient workflows difficult. That is to say, the vast majority of n-flows found do not appear to complete a task. Also, the most common n-flows found are redundant, in that they are often a subset of another common workflow. Additionally, the most frequently occurring n-flows consist of tools that nearly everybody uses, such as saving, moving the cursor, and selecting text, which means they would not be as effective for recommendations. In a recommendation system, workflows would need to be recommended that are necessary, efficient, and accomplish a task.

4.2 Future Work

Building on the results presented here, I can use the idea of recommending workflows to create a functioning recommendation system based on the algorithms mentioned in Section 2. Theoretically, it could be possible to go through these n-flows and recommend only those that meet the definition of a workflow, as defined earlier. Then, a recommendation system could be created that only recommends relevant workflows and tools to users. Finally, a study could be carried out to determine the effectiveness of recommending workflows. Only then could the insight of this paper be confirmed: that learning more efficient workflows makes software developers more efficient.

5. REFERENCES

[1] P. Fournier-Viger, A. Gomariz, T. Gueniche, E. Mwamikazi, and R. Thomas. TKS: Efficient mining of top-k sequential patterns. In *ADMA (1)*, pages 109–120, 2013.

[2] F. Linton, D. Joy, H. Schaefer, and A. Charron. OWL: A recommender system for organization-wide learning. *Educational Technology & Society*, 3(1):62–76, 2000.

[3] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice. CommunityCommands: Command recommendations for software applications. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, pages 193–202, 2009.

[4] E. Murphy-Hill, R. Jiresal, and G. C. Murphy. Improving software developers' fluency by recommending development environment commands. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 42:1–42:11, 2012.

[5] P. Viriyakattiyaporn and G. C. Murphy. Improving program navigation with an active help system. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '10, pages 27–41, 2010.