Developer Support and Assistant

Xiaozhe Yao

November 3, 2019

With the development of modern software engineering, there appear more and more tools to help developers, from version control, automatic tests, debugging and reviewing code, etc. To better cooperate with other programmers and non-technical employees, programmers are usually supposed to master these skills well. However, these development tasks are highly complex and disparate and lead to context switches. Thus there are several approaches proposed to help developers utilize these tools.

(Bradley, Fritz, & Holmes, 2018) proposed a context-aware conversational assistant named Devy for developers. The most promising feature of this approach is context-aware so that it can track the contextual information, such as the "Current Project", "Active File" etc. With that information, it could reduce the effort when developers are interacting with the assistants. Though there are many general-purpose conversational assistants and developer support tool, Devy seems to be the first to combine these and try to reduce the complexity of software development workflow. The experiments showed that industrial developers can perform basic tasks with Devy and it has enhanced their professional workflow.

However, as far as I can tell, there are some key improvements to better address the problem.

First, As mentioned by the authors, the predominant concern mentioned is the performance of the voice interfaces in open office environments. A more comprehensive experiment, which is conducted in the open office environments should be conducted to accommodate such concerns. Meanwhile, the experiment should also cover those co-workers who are in the same office but do not use the conversational assistants. It is necessary because sometimes people might feel weird when others are speaking loudly in a quiet office. Besides, when two or more developers are using Devy at the same time, would Devy run into conflicts and push the wrong files? I think current experiments are not sufficient to prove Devy is good enough for company developers, but the experiments successfully proved that Devy is suitable for indie developers and those who have a single office.

Second, Devy uses Amazon Alexa as voice input, which leads to at least 2 significant limitations. One is that it requires the users to use two names, "Alexa" and "Devy", which is cumbersome. The second is that it requires users to buy a device and configure it. I would suggest using a trivial microphone (e.g. Airpods) to configure Devy. It is not because of the cost or other commercial reasons, but because more users will adapt to it if they do not need to buy anything. With more developers and more voice data, it would possible to extend Devy to more complex cross-application workflows. Besides, with a near microphone, users could easily input their command in a lower voice, which might be helpful to reduce the chaos in the open office environment.

(Hoffswell, Satyanarayan, & Heer, 2018) tries to help developers debugging by introducing in situ code augmentation. Current debugging tools usually decouple the program state from the source code, which requires more attention and effort to connect them. To help developers reduce the switches between the

debugging tool and source code, the authors present a design space of embedded visualizations for time-varying variables. More specifically, they first proposed their design considerations, i.e. the augmentations must be comparable, unobtrusive and salient so that they can attract the developers' attention without detracting from their ability to perform current tasks. With these principles, they utilized the Vega editor to embed visualizations for Vega variables. Then they conduct an experiment with 18 students and saw a significant positive effect.

There are many tries in the field of in situ transforms to help developers reduce context switches. For example, the Fira Code Font, which turns "!==" to "≠", i.e. from programming language symbols to mathematical symbols. Since most junior developers are quite familiar with mathematical symbols, it could indeed help developers reduce the switches between mathematical symbols to programming symbols. (Hoffswell, Satyanarayan, & Heer, 2018) showed a promising approach to help developers debug. Though the author tested it with Vega, I believe it is more potential in data-intensive development, for example deep learning:

In a general deep learning pipeline, developers are supposed to train the deep learning model and monitor the process of gradient descent. Currently, developers can only print them out and see it in bare numbers or switch between the source code and drawing toolkits to get an insight into how fast the gradient is descending. It is indeed hard to debug since there are so many hidden parameters and developers usually have to switch between different processes. Therefore the proposed approach might be really helpful in this scenario, and I would like to implement the approach into this field and conduct relevant experiments.

Another possible extending work might be to discover when to perform a context switch. As is known to developers, a decoupled interface will definitely have more potential in providing more relevant information. It is a good try to visualize code in situ for some cases (especially when the data are irrelevant to others), but sometimes developers may prefer to view it in an independent window. It might be a good topic to find out when to choose a single window and when to visualize it in situ.

Both of these papers try to reduce the context switches, one from a different source of development workflow, and the other one from debugging information and source code. They used different approaches, by conversational assistants and in situ visualizations. By far we could conclude that reducing context switches is a hot topic in the field of developer support and assistant. However, I believe that we also need to try to improve the value of a specific context switch. For example, if we could provide more comprehensive information on a decoupled interface, though the developers need to perform a context switch, they may also benefit since they could quickly resolve the problems in such an interface.

In all, we should never tolerate useless and low-value context switches. Then we could try to reduce the number of context switches and improve the value of them at the same time. By improving the value of context switches, developers can also reduce the number of switches since they could rapidly resolve the problems they are facing.

References

Bradley, N. C., Fritz, T., & Holmes, R. (2018). Context-aware conversational developer assistants. In *Proceedings of the 40th International Conference on Software Engineering - ICSE '18*. ACM Press. https://doi.org/10.1145/3180155.3180238

Hoffswell, J., Satyanarayan, A., & Heer, J. (2018). Augmenting Code with In Situ Visualizations to Aid Program Understanding. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. ACM Press. https://doi.org/10.1145/3173574.3174106