Improving Code Completion

by Myroslava Romaniuk

Mentors: Marcus Denker, Stephane Ducasse, Oleksandr Zaytsev, Guillermo Polito

Introduction

In the ten years since its creation, the Pharo programming environment has evolved greatly: it now has Github support and many cool tools; it's faster, better and more convenient to use both for people who have been familiar with Smalltalk for decades and for complete newbies. However, there's one thing that is behind in improvements -- code completion.

The existing one is ancient and most people have no idea how it works and what's in there. And hence, even though everyone using Pharo wants it improved, hardly anyone has the patience to dive deep and spend hours and hours understanding the implementation and what can be done to change it for the better. Last year I did a summer internship where I already worked on code completion in Pharo. In general, with the help from supervisors, I managed to create a separate tool that already performed better than the existing code completion. Now, there are essentially two things to be done: improve the completion accuracy and effectiveness even further, and to actually port it into and make it fully workable in the Pharo environment itself. I will elaborate on the work already done and the proposed solution in the next section.

Proposed solution

To get to the things I want to implement, I will briefly go over what has already been done.

The initial idea was to improve the existing code completion but it quickly became evident that everything needed to be scrapped and that it was easier to build a new code completion model from scratch. Although eventually we decided that there were still some aspects of the existing one that were worth keeping, mainly the part that connected the implementation with the rest of the environment and made everything work.

What was done

- Developed a completion engine that uses AST (abstract syntax tree) nodes information to complete strings it receives; it works but can be improved
- Created a matcher that uses a visitor to collect all the fitting completion options based on different conditions (i.e. whether it's a literal, message or variable node)
- Started work on the sorter that would only propose completion suggestion in a certain order (so far, alphabetical and reverse-alphabetical)
- Implemented a fairly extensive testing functionality to test the tool
- Started porting the tool inside the Pharo programming environment

Next steps (during GSoC)

- **(1) testing and CI** - rewriting / writing more tests and setting up the CI to make sure that everything that is already implemented works 100%, as well as write new tests for the functionality to be added during GSoC, as per the TDD approach.
- **(2) porting the engine** - fully port completion engine inside Pharo. It has already been made possible to enable our completion controller in the settings, but most of the functionality has been borrowed from the existing completion models just for testing, and so it will have to be rewritten / greatly improved. There are also some challenges in making our code completion work for both the playground and the editor, because those functionalities and how our completion engine behaves in them have to be addressed separately. This will also be done as part of this step.
- **(3) statistical sorting** - extend the sorter functionality by implementing several more sorting strategies, i.e. sorting by recently modified methods and by the hierarchically closer methods (first methods of the class are displayed, then the superclass, etc).
- **(4) improving completion engine** - implementing additional features in order to enhance the code completion even further (e.g. support for code in braces, incorrect punctuation, etc).
- **(5) type inference** - right now we only do limited type inference, i.e. we can only tell the type when there's an assignment in a method with a known type. The solution would be to implement a visitor that annotates the AST with type information, as well as make type inference pluggable so that better implementations can be added later.

Timeline

May 6 - May 26 :
- Community bonding (getting starter feedback, etc)
- Specifying all the requirements
- Starting coding work on **(1) testing and CI**, due to exams in early June

May 27 - June 23 :
- Exams for the first two weeks, a bit less time spent coding
- Continue **(1) testing and CI**
- **(2) porting the engine** (rewriting models, updating controller architecture, etc.)
- Start gathering user/dev feedback

June 24 - June 28 :
- Evaluations

July 1 - July 21 :
- **(3) statistical sorting** (recently modified + hierarchically closer)
- **(4) improving completion engine**

July 22 - July 26 :
- Evaluations

July 29 - August 18 :
- **(5) type inference**
- Complete documentation and tests

- If time, add more features for better completion from user/dev feedback

August 19 - August 26 :

- Finalising the project and evaluations

Deliverables

- A code completion engine that performs better than the existing one
- Well-tailored and comprehensive UI
- Well-tested and well-documented completion tool that can be scaled if needed

Benefits for the community

The longer you do programming, the more you understand that having a nice completion is vital -- a good implementation enhances the coding experience. And so I believe that with this project the community will get a better and more efficient code completion that will improve their day to day use of Pharo, but which also will be well-documented and well-tested. That will make it understandable to people who try to have a look at it, and if the need arises, they will be able to improve, modify and/or scale it without my help.

Related work

A company working in Pharo called **feenk** has been developing their own code completion. While they for sure are doing some interesting things and it would be nice to sync with them, it has no bearing on the Pharo programming environment itself as their version of code completion is specifically tailored to the tool they work on, GToolkit (Glamorous Toolkit is a moldable IDE for Pharo).

About me

Github: https://github.com/myroslavarm
Email: romaniuk@ucu.edu.ua
Linkedin: https://www.linkedin.com/in/myroslavarm/
Medium: https://medium.com/@myroslavarm

Education

- Third-year Computer Science student at the Ukrainian Catholic University

My experience with Pharo

- Started working with Pharo 3 years ago
- Fixed bugs in Pharo 6 and 7
- Worked with Yuriy Tymchuk on Renraku (improving rules of quality assistant)
- Successfully completed Google Summer of Code 2017 with Pharo Consortium

- Organized and managed Pharo Club at my university - taught Pharo to younger students
- Student volunteer at ESUG 2017 and 2018
- Did the first steps towards improving the code completion engine in Pharo during my 2 months internship in the RMoD team at INRIA Lille in the summer of 2018: https://github.com/myroslavarm/Experimental-Completion

I like using Pharo both because of an opportunity to do cool things in a way most other languages/environments don't provide and because of the community that works on and in it, and so I am determined to keep trying to make it better, more convenient to use for experienced users, and easier to use for newbies. I strongly feel that working in software quality and open source helps me in my personal and professional development, and I hope that the things I do in Pharo can help other people, too.

Why am I good for this project?

- have several years of experience programming in Pharo
- already did a big part of the code completion project last summer, so now don't have to start from scratch but can continue with already some knowledge of the project
- have participated in the Pharo dev/user community and myself have been a user of Pharo long enough that I have an idea of where I want this project to be heading to have the maximal positive outcome