Augmenting software development with information scripting

Master Thesis Description

Lukas Vogel
luvogel@student.ethz.ch

May 26, 2015

1 Introduction

Today's large software projects are associated with a lot of information. The program code itself is perhaps the most important information source. Additionally issue trackers, version control systems, debuggers, analysis tools, and documentation, etc. provide invaluable information to a developer. Engineers frequently face questions which require a combination of information from multiple sources [1] to answer. Yet existing tools are often not optimal for such use cases.

Consider an imaginary developer Bob, who is debugging a new regression. Bob updated his code version with the version control system. He finds the updated code crashes somewhere after the current breakpoint in the method `addTracks()`. To find the cause Bob would like to know what changes occurred in the callgraph of the method `addTracks()` during the last week. Bob thus has to answer the more abstract query: "Find recent changes in the callgraph of function x". To answer this query a developer needs to find all functions that lie in the call graph of the function x. With the version control system the developer can determine which code has recently been changed. Those two information fragments then have to be manually combined to answer the initial query. This pattern where a combination of multiple subqueries is needed to answer a question is common in software development. Both LaToza and Fritz [1, 2] identify similar questions which are shown to be cumbersome to answer in existing tools. While existing work solves some issues, there are still open questions and there is room for improvement.

In this project we aim to design an information scripting framework that allows a user to query, compose, and visualize information from different sources. We believe that such a system can help developers to answer many of the raised questions. The framework should be integrated in Envision, an IDE prototype for object-oriented languages [3]. Envision features a visual structured editor based on a flexible visualization framework that scales to large programs.

2 Design inspiration

For the information scripting framework we pursue an approach which is heavily inspired by the Unix command line and its power. We aim to design the framework in four layers:

- **The exchange medium** is the data structure that all other layers use to communicate with each other. In Unix the exchange medium is lines of text.
- **The information sources** are sources of information such as a version control system or the source code. This layer is analogous to Unix programs that produce data without input, for example `ls`.
- **Scripts** are used to combine, filter, sort, etc. the data of the exchange medium. Unix supports this with pipes, streams, and scripts.
- **Visualizations** should present the output of a script in a meaningful way. In Unix this is usually plain text, either displayed on the screen or saved as a file.

A somewhat similar question to the one we presented in the introduction is "Print the last three commit ids for all source files which contain the word label". In Unix this can be answered with the following command:

```
grep -l "label" *.c | xargs -L 1 git log -3 --format=%H --
```

This line shows how a user can answer a query with a single line of combined commands. We want to bring this power and simplicity to an IDE and source code.

3 Related work

Fritz and Murphy [1] identify a set of questions developers ask routinely and aim to simplify the answering of those questions. They define various information fragments as bits of information from various sources, e.g. source code, change sets, etc. They show that with an information fragment model which supports composing and presenting information fragments developers are able to answer most raised questions with ease. While Fritz's solution is specifically designed around a set of questions, with our scripting approach it should be possible to cover more cases.

Storey et al. [4] observe that for a programming task often various locations of a software project are involved. A location in this sense can be a source location, a wiki entry, a comment, etc. By creating a waypoint system they aim to group important locations for a task. With a prototype implementation they gathered mostly encouraging feedback. In our work we aim to generalize this idea, where tags are just one source of information. The tag information could be combined with other information which can be even more useful. For example a combination of tags and version control information could be used to get updated information for a certain task.

Schiller and Lucia [5] describe a plug-in ecosystem that should enable users to combine various plug-ins inside an integrated development environment. They

formalize this system using a polymorhic lambda calculus. For the interaction between the plug-ins they define various data models. The formalism from the lambda calculus assures that nonsensical plug-in combinations are impossible. The authors implemented their approach in a prototype called Cupid. Cupid uses a combination of clickable interfaces with textual input to create information queries. To display the results a separate formatting rule has to be applied. In our work we aim to create a solution which allows a user to easily combine the querying and visualizations in a single command or script. Furthermore the visual environment of Envision provides new opportunities on how to meaningfully display the queried data.

LaToza and Myers [2, 6] identify questions which developers frequently ask and which are time consuming and hard to answer. They suggest that these questions should be taken into consideration when designing development tools. Especially reachability questions [6] are identified to play an important role in software engineering. The authors designed the tool Reacher [7] which can help answer call-graph related questions faster than traditional tools. For our work the analysis of frequently asked questions and the tool Reacher will be an important source of inspiration for the design process.

4 Core tasks

This project includes several core tasks. At first we will explore existing work. Then our work focuses on designing and implementing a four-layered information scripting system. This system should gradually be integrated in Envision.

4.1 Analysis of existing work

There exist various approaches on combining of data from different information sources. We presented a small selection in section 3. We will analyze the existing work and identify advantages and disadvantages. This analysis will be used to guide our work.

4.2 Data exchange layer

To make it possible to query and compose information from different data sources we need a uniform data format, which is understood by all data sources. The format should be able to express complex relationships between data. We aim to explore a graph-based approach for this data format, as a simple list is insufficient to encode all relevant information.

Consider a query `callgraph` which returns the callgraph of a method. The data format should be able to encode such information. Another example is the data flow of a variable which is passed as an argument to a method.

4.3 Information source layer

The information source layer provides a way to access information from various sources. Some potential sources are the source code, version control, compiler, issue tracker, tags, etc. Envision should at least feature the source code and tags as data sources. A tag is a piece of information attached to a source location

similar to what Storey et al. proposes [4]. From the implementation perspective this layer is just an interface over the existing IDE, that can query the relevant information and return it in the unified data format.

An information source more specifically should allow a programmer to request certain information by entering a command with arguments. The requested data should be returned in the data exchange format as specified in section 4.2.

A simple query to the source code data source could be `calls draw`, which should return all locations that call the `draw` method. Another example of a simple query to the version control system could be `changed recently`, which should return all locations that have recently changed.

4.4 Scripts layer

To enable more complex queries we plan to provide data source independent tools which work on the unified data format. We plan to provide tools for combining, filtering, piping, and sorting information in the unified data format. A single command, or a combination of commands can also be saved as a script and then serve as a proper data source.

If we reconsider the two examples from the information source layer, a script should make it possible to combine the queries, such that a user can get all code regions which call the method `draw` and have recently been changed.

4.5 Visualization layer

The visualization layer should provide visualization primitives which can be used to visualize the unified data format. At the very least we should have the following visualization primitives:

- **Highlights** To visualize certain code regions of interest. For example this could be used to highlight code regions which have recently been changed.
- **Arrows** To visualize connections or dependencies between different parts of the data. Arrows can, for example, be used to visualize a call graph.

The idea is that per default all output of the commands to the introduced system are visualized by this layer.

5 Possible extensions

Here we list some possible extensions to the core tasks.

5.1 Advanced visualizations

Investigate and implement more visualizations. We list two specific examples here but the list could be extended depending on the progress with other tasks.

5.1.1 Heat maps

A heat map is an overlay which colors different regions of the code in different colors depending on some heat measure. In a temperature heat map the color red means hot and blue cold, we could use a similar notion or experiment with a different paradigm.

The heat map should support visualizing code regions which have been recently or frequently changed. Additionally it should support showing which authors are active in which regions of the code. This could help developers which are new in a project to find the person which they have to talk to if they encounter a problem in a certain location of the project.

5.1.2 Data flow

In programs, data can flow through various paths. In some cases it is important to know where the data of certain variables flows through. We want a visualization of this flow to help developers in certain questions. Consider the question "Can I change the definition of this variable without getting code conflicts with my coworkers?", a data flow visualization and a map on what the coworkers are working on, might help to answer such a question.

5.2 Modifying Commands

Instead of only querying data we would like to extend the proposed system to support modifying data through commands. Such a command could, for example, make it possible to change all variables of type `std::string` to variables of type `QString`.

Such a change might break something during compilation. Thus we would like that modifying commands can automatically incorporate feedback from other sources, e.g. the compiler.

That mechanism would enable refactoring actions where instead of updating all references when renaming a field, the compiler is used as a data source to suggest which references have to be renamed.

5.3 More information sources

To showcase the ability of our system we should support more data sources.

We aim to use the version control system as a data source. It could be used to query recent changes, frequent authors for certain locations, etc.

Envision recently gained a debugging plug-in, which can compile, run, and debug code. This could be extended to serve as an additional data source. The compiler could be used in combination with a modifying command for refactoring actions as explained in the previous task.

6 Schedule

A tentative time schedule for the project

Task	Start date	Time
Explore related work	June 1, 2015	2 weeks
Design unified data format (layer 1)	June 15, 2015	2 weeks
Implement data sources (layer 2) & data format (layer 1)	June 29, 2015	3 weeks
Introduce script capabilities (layer 3)	July 20, 2015	3 weeks
Introduce visualization primitives (layer 4)	August 10, 2015	2 week
Refine unified data format (layer 1)	August 24, 2015	2 weeks
Finalize core tasks	September 7, 2015	2 weeks
Extensions	September 21, 2015	6 weeks
Write-up & finalizing	November 2, 2015	4 weeks
End	December 1, 2015	

References

[1] Thomas Fritz and Gail C. Murphy. Using information fragments to answer the questions developers ask. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 175–184, New York, NY, USA, 2010. ACM.

[2] Thomas D. LaToza and Brad A. Myers. Hard-to-answer questions about code. In *Evaluation and Usability of Programming Languages and Tools*, PLATEAU '10, pages 8:1–8:6, New York, NY, USA, 2010. ACM.

[3] D. Asenov and P. Müller. Envision: A fast and flexible visual code editor with fluid interactions (overview). In *Visual Languages and Human-Centric Computing (VL/HCC)*, pages 9–12, 2014.

[4] Margaret-Anne Storey, Li-Te Cheng, Ian Bull, and Peter Rigby. Waypointing and social tagging to support program navigation. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1367–1372, New York, NY, USA, 2006. ACM.

[5] Todd W. Schiller and Brandon Lucia. Playing cupid: The ide as a matchmaker for plug-ins. In *Proceedings of the Second International Workshop on Developing Tools As Plug-Ins*, TOPI '12, pages 1–6, Piscataway, NJ, USA, 2012. IEEE Press.

[6] Thomas D. LaToza and Brad A. Myers. Developers ask reachability questions. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 185–194, New York, NY, USA, 2010. ACM.

[7] T.D. LaToza and B.A. Myers. Visualizing call graphs. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 117–124, Sept 2011.