Using Mountain Visualizations for Orientation in Source Code

Jan Nonnen

University of Bonn
Computer Science III
Bonn, Germany

nonnen@cs.uni-bonn.de

Abstract

In source code navigation developers memorizes navigation features for orientation even if they are hardly aware of it. Furthermore, a majority of development time is spent on code navigation. We propose a novel visualization of source code indentation to provide additional landmarks for orientation. This visualization uses the human perception of landmarks in natural landscapes as orientation objects.

Keywords: software visualization, indentation, program comprehension, landmarks, orientation, code navigation

1 Introduction

One of the developer activities performed most of the time is code navigation. Ko et al. [5] reported that programmers spent about 35% of their time with navigation in source code. During navigation it is important for the developer to memorize navigation features to be able to later find the way back to the visited locations. The so called spatial memory is then accumulated into a cognitive map [8]. This map is based on two elements: layout orientation and landmark orientation.

Elias [3] defined landmarks to be objects that have a distinctive shape in regards to their surroundings. She emphasized that humans rely on these landmarks for orientation and that we perceive them automatically through vision. In source code the shape of the source code could be used to identify landmarks for orientation. In our approach, we want to use the indentation of the source code to create a visualization of this as explicit landmarks for orientation.

The effect of indentation on program comprehension is often overlooked. In 1983, Miara et al. [7] observed in a user study on the Pascal programming language that the level of indentation had a significant effect on program comprehension. In their study, the best comprehension was achieved with an indentation of 2 or 4 spaces. They observed that more indentation lead to harder program comprehension.

The Sun Java coding guidelines have only a short note mentioning that one should use 4 spaces as a replacement for one tab and a few additional rules for indenting parameters [9]. We studied the Qualitas Corpus [10] release 20101126r and found that the majority of source code lines used only spaces as indentation (78%). In this corpus, 18% of the lines where indented with tabs only and 4% used a mixture of both. Most lines indented with spaces had an indentation level of a multiplicity of 4, thus satisfying the Sun Java coding guidelines. Interestingly, the corpus contained every indentation level up to 152 spaces. Furthermore, the indentation levels above 140 can be traced back to parameter indentation of nested method calls.

Importance of indentation was mentioned also by Martin [6, p.88]. He explained that a source file is rather a hierarchy and that we use indentation to make it visible. To show this hierarchy usually methods are indented one level, loop bodies have a higher indentation level than the surrounding block. This allows to see parts of the structure already by indentation: *"Your eye can rapidly discern the structure of the indented file. You can almost instantly spot the variables, constructors, accessors, and methods."*[6, p.88].

DeLine et al. [1] observed an effect of indentation for orientation and navigation. They suspected that the source code outline implicitly helped the participants of their study to navigate faster.

Hindle et al. [4] studied the correlation between the indentation of revisions and classical complexity metrics such as Halstead's and McCabe's. One of their findings was that indentation is better suited for measuring nesting than the Halstead metric.

Based on the findings presented above, indentation seems to be a good indicator for measuring nesting and the shape of the source code seems to help orientation.

2 The Mountain Outline

We propose a novel visualization of the source code indentation to assist developers in source code orientation and navigation. The visualization is created by drawing the contour of indentation. Then this re-

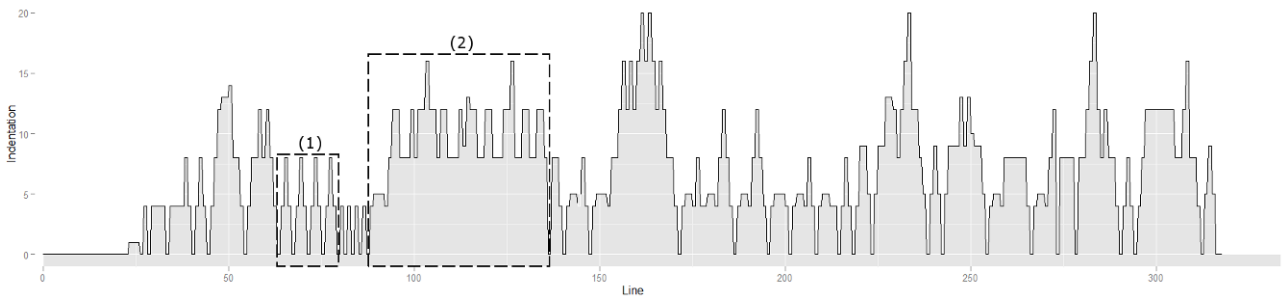

Figure 1: The mountain outline for the `BaseTestRunner` class from JUnit 4. (1) shows the usual pattern for simple methods (e.g. getter/setter), (2) highlights a problematic long and complex method.

sulting polygonal chain is rotated by 90 degrees anticlockwise. We call this rotated indentation contour the **mountain outline**, because it reminds of a landscape outline of a mountain range [1] .

Figure 1 shows a visualization using this mountain outline of the `BaseTestRunner` class in JUnit 4. One is able to directly identify simple methods (1) and complex nested methods (2). The complex method is over 50 lines long and contains 3 nested blocks. For a developer this method would be a good starting point for refactoring opportunities and for understanding the complete class. One can also recognize a set of shorter methods in the lines 170 till 230 between two more complex methods.

A prototype for Eclipse provides this visualization as an additional view that is connected to an editor. One can navigate to a specific line by using the mountain visualization. Further, the current selected line is highlighted in the visualization.

3 Related Work

The SeeSoft tool by Eick et al.[2] provides a visualization of source code lines. They use color to display line oriented software metrics, e.g. time of change.

A thumbnail visualization of the source code was presented by DeLine et al. [1]. In contrast to our approach, they use the complete source code including the content. One needs to study whether the use of indentation provides a shape that is more stable over time.

4 Future Work

A smaller user study of the mountain outline prototype is planned for April 2013 and a larger user study for June. In this user study we plan to plan to address the question, whether or not the prototype reduces navigation time and source code landmarks are remembered by the participants.

In the future we plan to extend the mountain contour to show the contour last seen(by the user) and the current source code outline. This would allow to recognize old landmarks and a relocation of those on the new contour.

The mountain outline uses the idea of humans easier perceiving landmarks for orientation in a horizontal landscape. One needs to validate, if vertically aligned source code and translating this into a horizontal visualization has a negative impact on usability.

One drawback of the outline are its limitations when dealing with good quality code. Good code quality with short and simple methods results in a smaller mountain outline and less outstanding landmarks. In the future we plan to evaluate, if a mountain outline in this case still improves orientation.

[1] This idea was also mentioned by R. C. Martin in the Clean Coder videos in episode 3, http://cleancoders.com/

References

[1] R. DeLine, M. Czerwinski, B. Meyers, G. Venolia, S. M. Drucker, and G. G. Robertson. Code thumbnails: Using spatial memory to navigate source code. In *VL/HCC*, pages 11–18, 2006.

[2] S. G. Eick, J. L. Steffen, and E. E. Sumner Jr. Seesoft—A tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, Nov. 1992.

[3] B. Elias. *Extraktion von Landmarken für die Navigation.* PhD thesis, Universität Hannover, 2006.

[4] A. Hindle, M. W. Godfrey, and R. C. Holt. Reading beside the lines: Using indentation to rank revisions by complexity. In *Science of Computer Programming*, volume 74, pages 414–429, May 2009.

[5] A. J. Ko, H. H. Aung, and B. A. Myers. Eliciting design requirements for maintenance-oriented IDEs: a detailed study of corrective and perfective maintenance tasks. In *ICSE*, pages 126–135. ACM, 2005.

[6] R. C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship.* Prentice Hall International, 2008.

[7] R. J. Miara, J. A. Musselman, J. A. Navarro, and B. Shneiderman. Program indentation and comprehensibility. *Commun. ACM*, 26(11):861–867, 1983.

[8] J. O'Keefe and L. Nadel. *The Hippocampus as a Cognitive Map.* Oxford University Press, 1978.

[9] Sun. Java Code Conventions, 1997.

[10] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, pages 336–345, Dec. 2010.