Eurographics Conference on Visualization (EuroVis) 2016
E. Bertini, N. Elmqvist, and T. Wischgoll
(Guest Editors)

Short Paper

Using icicle trees to encode the hierarchical structure of source code

I. Bacher[1], B. Mac Namee[2], J. D. Kelleher[1]

[1]Dublin Institute of Technology, Dublin, Ireland
[2]University College Dublin, Dublin, Ireland

Abstract

This paper presents a study which evaluates the use of a tree visualisation (icicle tree) to encode the hierarchical structure of source code. The tree visualisation was combined with a source code editor in order to function as a compact overview to facilitate the process of comprehending the global structure of a source code document. Results from our study show that providing an overview visualisation led to an increase in accuracy and a decrease in completion time when participants performed counting tasks. However, in locating tasks, the presence of the visualisation led to a decrease in participants' performance.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications—

1. Introduction

Efficiently understanding source code is an important problem in software engineering [Tel14]. Indeed, previous studies have shown that understanding source code accounts for more than half of software development effort [Sta84, Cor89]. Source code contains many types of relationships and hierarchies, such as the dependencies of packages, the package-class-method-statement hierarchy, or hierarchies of data structures. Given these particular properties and the high complexity of source code it is reasonable to consider how visualisation can facilitate source code comprehension, as it is recognised that visual representations tap into the capabilities of the powerful and highly parallel human visual system [Moo09].

A source code editor is a text editor designed specifically for the writing and editing of source code, which makes it a fundamental tool for software developers. These editors typically have features, such as syntax highlighting and pretty printing, which facilitate the process of viewing, reading, and comprehending source code. This is done by making the structural and syntactical composition of a source code document more visible, through the use of indentation, spaces, line-breaks, colour, and font-face. Previous studies have shown that the typographic appearance of source code [BM90] and the indentation style used within source code [MMNS83] can influence the speed and accuracy of program comprehension. However, source code documents are frequently too large to be displayed on a single screen. This introduces the need for scrolling, which can cause a cognitive burden for the user who must mentally assimilate the overall structure of the information space and their location within it [CKB09]. While source code editors provide users with an overview of the package-class-method-statement hierarchy in a software project, they fall short in providing the user with an overview of the hierarchical structure of a source code document, which at the same time encodes the users current location within the information space. Thus, exacerbating the cognitive burden introduced by scrolling.

Overview is a frequently used notion and design goal in information visualisation research and practice. However, it is difficult to find a consensus on what an overview is. For this work we define overview with the help of a taxonomy created by Hornbaek and Hertzum [HH11], as it incorporates the most important aspects of how the notion of overview is used in information visualisation. Using the taxonomy [HH11, p. 511] we define an overview as *an awareness of the structure of an information space, acquired by information reception throughout a task, useful for understanding with good performance, and provided by a semantically shrunken dynamic visualisation*.

Our research presents a study that evaluates the use of a tree visualisation that provides the viewer with an overview of the hierarchical structure of a source code document. The goal of the overview visualisation is to provide a compact representation of a source code document, which in turn should support software developers during software development tasks. The main contributions of this work are: (1) The use of a tree visualisation to provide an overview of the hierarchical structure of a source code document. (2) An evaluation of the visualisations ability to aid source code comprehension by means of a between group experiment.

2. Related work

Hierarchies are a common and powerful tool for structuring relational information. A tree structure or tree diagram is an appropriate way of representing the hierarchical nature of information in

graphical form. Many techniques have been developed for the display of hierarchically structured information. Treevis.net [Sch11], a web based survey and overview, includes more than 180 different tree visualisation techniques. These techniques can be split into two categories: implicit and explicit tree visualisations [SHS11]. Explicit tree visualisations, such as the node link diagram (Figure 1), show parent-child relations as straight lines, arcs, or curves. Implicit tree visualisation, such as tree-maps (Figure 2) and icicle trees (Figure 4), represent parent-child relations by the use of juxtapositioning, which can lead to a more space efficient layout. As the aim of this work is to provide a compact overview visualisation of the hierarchical structure of a source code document, we will focus on implicit tree visualisations.

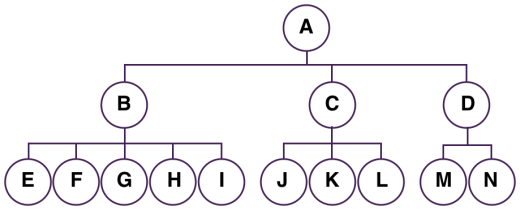


Figure 1: *Node-link diagram*

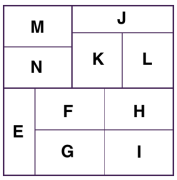


Figure 2: *Tree-map*

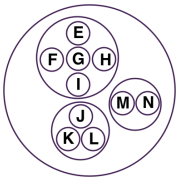


Figure 3: *Cicular tree-map*

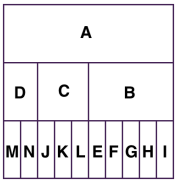


Figure 4: *Icicle tree*

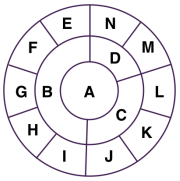


Figure 5: *Sunburst diagram*

Tree-maps [JS91] (Figure 2) provide an overview of an entire hierarchy and are generated by recursively slicing the available display space into smaller boxes for each level of the hierarchy, using horizontal and vertical slicing algorithms. Hierarchical relations are encoded by the use of nesting, thus all elements of a hierarchy are able to be displayed in the available display space. Circular tree-maps [WWDW06] (Figure 3) are an extension of the tree-maps. Instead of boxes, the technique uses nested circles, which makes it easier to see groupings and hierarchical organisation. Sunburst diagrams [SCGM00] (Figure 5) rely on a circular or radial display to represent hierarchy. Nested discs or portions of disks are used to compactly visualise each level of a hierarchy, where the deepest element in the hierarchy is located the furthest from the center. Icicle trees (adjacency diagrams) [HBO10] (Figure 4) are largely similar to node-link diagrams, but instead of a node-link construct, they employ an adjacency-area method where a series of juxtaposed rectangles indicate rank. Icicle trees are able to adopt either a vertical or horizontal layout, thus making them highly adaptive to space and layout constraints.

While, to the best of our knowledge, no experiments have been done looking explicitly at the use of tree visualisations in the context of providing an overview of the hierarchical structure of a source code document, several experiments have been done in the area of information visualisation to evaluate the usability of various tree visualisation techniques.

Cawthon and Moere [CM07] investigated the results of an online survey of 285 participants, measuring both perceived aesthetic as well as the efficiency and effectiveness of retrieval tasks across a set of 11 different tree visualisation techniques. Their findings show that sunburst diagrams and icicle trees performed highest in terms of correct responses. Stasko et al. [SCGM00] conducted two empirical studies of two tree visualisation techniques for depicting file hierarchies. These techniques were tree-maps and sunburst diagrams. In both studies participants were given a series of tasks where all of the tasks required a participant to find or identify a particular file or directory, or to make a comparison of two files or directories. Results indicated that the sunburst method aided task performance more frequently, both in correctness and in time, particularly for larger hierarchies. This was due to the fact that the sunburst method explicitly portrayed the structure of a hierarchy, which appeared to be a primary contributor to its performance. Overall, participants in both studies preferred the sunburst method. Barlow and Neville [BN01] conducted a study in the context of decision tree analysis that compared node-link diagrams to three alternative tree visualisations: icicle tree, tree-map, and tree-ring. The study evaluated the ability of tree visualisations to communicate a decisions tree's topology and support for common decision tree analysis tasks. Decision trees depict rules for dividing data into groups and the predictive model created by the decision tree suggests which variables and relationships are important. Overall the tree-map was uniformly disliked by the participants and their performance while using it was worse than with the other tree visualisations. Results suggest that the icicle tree is equivalent to or better than the node-link diagram and tree-ring in most tasks. Moreover, the authors also mention that certain tasks favoured the icicle tree, while others the tree-ring.

The studies described above illustrate that many different tree visualisation techniques can be used to depict the hierarchical structure of an information space. Although tree-maps are well known, this method falls short in conveying the global structure of a hierarchy as non-leaf nodes are not shown [War13, p. 229]. Thus techniques, such as sunburst or icicle trees, which portray the structure of a hierarchy through juxtaposition, are more suitable for overview visualisations. For this work, we explore the use of an icicle tree for encoding the hierarchical structure of source code, as we believe this technique best satisfies the design rational stated in section 3.

3. Overview visualisation approach

Figure 6 illustrates our prototype interface, which is composed of an icicle tree and a source code editor. Nodes within the icicle tree

encode the hierarchical structure of the HTML document, located in the source code editor, and are represented as graphic primitives. Parent-child relationships are encoded by horizontal adjacency, meaning that child nodes are placed to the right of their parent. Depending on which structural element the text cursor is located within the source code editor, the corresponding node within the icicle tree is highlighted. The icicle tree's layout is calculated using a space-filling algorithm, in order to use the available display space and avoid scrolling. However, this introduces a limitation that at a certain number of nodes, the visualisation becomes too packed to comprehend. This limitation was taken into consideration, as our experiment was explicitly designed to evaluate the performance of the icicle tree for different levels of source code complexity, which is measured using metrics such as lines of code and total number of nodes. Further details can be found in section 4.

Figure 6: *Prototype interface composed of an overview visualisation and source code editor*

In order to help visualisation designers and researchers, several design guidelines and principles have been proposed [Wat05, PBG98, PdQ*06, CJHS95]. While all of these design guidelines and principles are applicable, we felt that a subset correspond particularly well for the purpose of visualising hierarchical relations within source code. These can be described as follows: Language specific, Order adjacency, Awareness, Interaction, and Scalability

Language specific [CJHS95]: Many different types of computer languages exist such as Java, C#, JavaScript, HTML, and XML, each containing their own set of relations and hierarchies. For example, Java contains a Package-File-Class-Method-Statement hierarchy, while JavaScript is more flexible and does not impose this strict hierarchical structure. The goal of the overview visualisation is to provide the viewer with an overview of the hierarchical structure of a source code document. Hence, depending on which computer language the source code is written in, different hierarchical structures can be extracted. For our work, HTML was the language of choice, because it imposes a very strict hierarchical structure.

Order Adjacency [Wat05]: In the context of source code, specifically HTML source code, the ordering of the structural elements is particularly important as it specifies the rendering of the document to some extent. Thus, this natural ordering should be preserved in order not to confuse the viewer and add mental burden. We felt that the icicle tree best fulfils this requirement as the it provides a natural projection of the ordering that aligns with the indentation of the code. Other implicit tree visualisations such as tree-maps, sunburst diagrams, and circular tree maps were also considered. However, the tree-map and circular tree map do not impose any type of ordering, and the sunburst technique uses a clockwise ordering. Hence, the icicle tree was the tree visualisation of choice.

Awareness [PdQ*06]: The goal of an overview visualisation is to provide a viewer with an overview of an information space, and show their current location within the information space. The icicle tree displayed in Figure 6 encodes the hierarchical structure of a source code document, hence, the viewer should be informed in which structural element they are currently located in within the information space. We identify the structural element that is currently of interest to the user based on the location of the cursor in the source code editor. The corresponding node is then highlighted in the icicle tree. Figure 6 illustrates this feature, as the cursor is currently located on line 136 in the source code editor, which contains a header element. Line number 136 to 138 are highlighted as this corresponds to the opening and closing part of the header element. The corresponding node is highlighted in the icicle tree.

Interaction [PBG98, PdQ*06]: Users should be able to use the overview visualisation to navigate though the source code located in the source code editor. The interaction should allow the user to interact with the overview visualisation to change the point of focus of the source code editor. Allowing the user to move rapidly to any location within the information space. In the current prototype (Figure 6), users are able to click on a node, within the icicle tree in order to navigate to the line of code which corresponds to the start of the structural element. However, in order to control for interaction effects in our evaluation experiments, no additional interactions were implemented. Future studies could include additional interaction options, which could be used to improve overall comprehension and understanding.

Scalability [PBG98]: In order to obtain a broad level of understanding of how many structural elements and hierarchical levels a typical HTML source code document is composed of, a HTML web-scraper was implemented. The web-scrapper queried the top 10.000 websites, ranked based on their monthly traffic. Results show that the median nesting level is 13 and the median number of nodes is 718. Our evaluation was designed to test the icicle tree across a range of source code documents with different levels of complexity with these median values included within the range of the complexity of our test documents.

4. Evaluation

The experiment followed a between group design and was implemented as a web application. Two versions of the application were hosted online, where the overview visualisation was present in one, while omitted in the other. 39 subjects participated in the experiment, of which 25 were currently enrolled in a computer science, web development, or software engineering course. 17 participants completed the experiment using the prototype with the visualisation, while 22 completed the experiment using the prototype without the visualisation.

The experiment consisted of four phases and took approximately

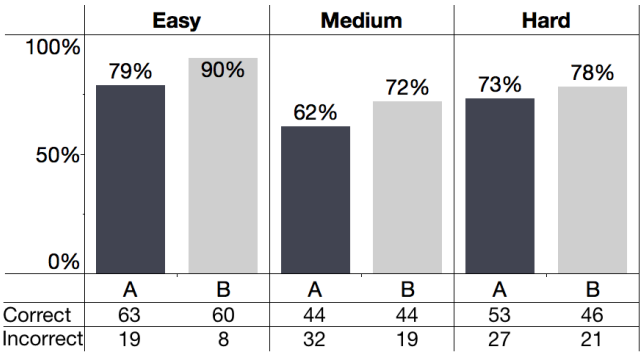
30 minutes to complete. During the first phase participants were presented with definitions for terms such as *node*, *child node* and *ancestor node* in order to become accustomed to the wording used within the experiment. In the second phase the participants were shown an image of the prototype, which included several annotations. The annotations were used to describe functionality such as a text search feature, or to describe the visualisation. In the third phase participants were given three source code documents and for each document they were asked to answer a set of questions. Each participant was presented questions and source code documents in random order. During this phase question accuracy and response time were gathered. The fourth phase consisted of a survey where the participants were asked to respond to general questions including programming experience, age group, and gender. Participants that completed the experiment with the visualisation were presented with two additional questions in the survey, which required them to rate the usefulness of the visualisation.

Three HTML source code documents were used and can be differentiated based on their complexity, which is measured using lines of code (loc), total nodes, and nesting levels. The *easy* source code document consisted of 47 loc, 8 nesting levels, and 30 nodes. The *medium* source code fragment consisted of 247 loc, 6 nesting levels, and 167 nodes. The *hard* source code fragment consisted of 2,787 loc, 13 nesting levels, and 1,605 nodes. Participants were asked to answer a series of questions for each source code document. The questions can be categorised into two types depending on the task the participant had to perform, counting or locating. Examples of the questions are: *How many child/descendant/leaf nodes does node "X" contain* and *what node is the closest common ancestor of nodes "X" and "Y"* (where, "X" and "Y" were replaced with unique identifiers that identified nodes within the source code using the HTML id attribute, e.g. id = "ZBHRB").

5. Results and discussion

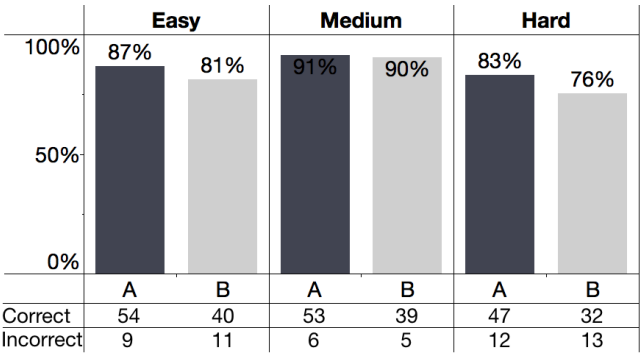
In the experiment 39 participants completed 819 questions where accuracy and response time was recorded for each question. Figures 7 and 8 depict the percentages of correct answers for counting and locating tasks corresponding to each source code fragment.

In regards to participant accuracy, an interesting pattern emerged when comparing questions based on task type. For questions that involved counting nodes, accuracy seemed to increase when the overview visualisation was present (Figure 7) for all three source code documents. This was surprising, as we only expected participants to have a higher accuracy score in the *medium* source code document due to the fact that the *easy* source code document only consisted of 30 nodes, and the *hard* source code document tested the limitations of the space-filling layout algorithm used for the overview visualisation. We suggest that this increase in accuracy is due to the fact that the visualisation fundamentally encodes structural elements of source code, thus making them easier to count. For questions which involved locating nodes, accuracy seemed to decrease when the overview visualisation was present (Figure 8). However, we believe that this could be improved by adding additional encodings to the overview visualisation, such as the use of colour for depicting previously visited nodes. In terms of question completion times, participants were generally faster using the



	Easy		Medium		Hard	
	A	B	A	B	A	B
Correct	63	60	44	44	53	46
Incorrect	19	8	32	19	27	21

Figure 7: *Counting task accuracy (A - without vis, B - with vis)*



	Easy		Medium		Hard	
	A	B	A	B	A	B
Correct	54	40	53	39	47	32
Incorrect	9	11	6	5	12	13

Figure 8: *Locating task accuracy (A - without vis, B - with vis)*

overview visualisation, however, this also led to a decrease in accuracy in certain questions. This is an interesting finding, as it suggests that using an inappropriate visualisation can actually have a detrimental effect on task performance, while at the same time giving participants false confidence that they are performing well. These results can be used as a first step towards illustrating the benefits and shortcomings of the use of tree visualisations in the context of providing an overview of the hierarchical structure of a source code document.

6. Conclusion

This work presented a study, which evaluated the use of a tree visualisation for encoding the hierarchical structure of source code, in order to facilitate source code comprehension. This is a first step towards providing empirical evidence on the usefulness of tree visualisations as overviews in combination with source code editors. Future work directions include evaluating similar tree visualisations, such as sunburst diagrams, using existing results as benchmarks. Additional interaction techniques such as zooming could also be added to the tree visualisation in order to support larger source code documents. Furthermore, different encodings could be added to the tree visualisation in order to make the viewer aware of previously visited nodes. This would also correspond with comments received from participants, as most did note that additional functionality would have been useful. The prototype is available on the Internet from `http://tiny.cc/no1ray`.

References

[BM90] BAECKER R. M., MARCUS A.: *Human factors and typography for more readable programs*. Addison-Wesley, Reading, Mass, 1990. 1

[BN01] BARLOW T., NEVILLE P.: A comparison of 2-D visualizations of hierarchies. In *IEEE Symposium on Information Visualization* (2001), IEEE, pp. 131–138. URL: http://doi.ieeecomputersociety.org/10.1109/INFVIS.2001.963290. 2

[CJHS95] CANT S., JEFFERY D. R., HENDERSON-SELLERS B.: A conceptual model of cognitive complexity of elements of the programming process. *Information and Software Technology 37*, 7 (1995), 351–362. 3

[CKB09] COCKBURN A., KARLSON A., BEDERSON B. B.: A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv. 41*, 1 (Jan. 2009), 2:1–2:31. URL: http://doi.acm.org/10.1145/1456650.1456652, doi:10.1145/1456650.1456652. 1

[CM07] CAWTHON N., MOERE A. V.: The effect of aesthetic on the usability of data visualization. In *Information Visualization, 2007. IV'07. 11th International Conference* (2007), IEEE, pp. 637–648. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4272047. 2

[Cor89] CORBI T. A.: Program understanding: Challenge for the 1990s. *IBM Systems Journal 28*, 2 (1989), 294–306. 1

[HBO10] HEER J., BOSTOCK M., OGIEVETSKY V.: A tour through the visualization zoo. *Commun. Acm 53*, 6 (2010), 59–67. 2

[HH11] HORNBÆK K., HERTZUM M.: The notion of overview in information visualization. *International Journal of Human-Computer Studies 69*, 7 (2011), 509–525. URL: http://linkinghub.elsevier.com/retrieve/pii/S1071581911000322, doi:10.1016/j.ijhcs.2011.02.007. 1

[JS91] JOHNSON B., SHNEIDERMAN B.: Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In *Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on* (1991), IEEE, pp. 284–291. URL: http://dx.doi.org/10.1109/VISUAL.1991.175815. 2

[MMNS83] MIARA R. J., MUSSELMAN J. A., NAVARRO J. A., SHNEIDERMAN B.: Program indentation and comprehensibility. *Communications of the ACM 26*, 11 (1983), 861–867. URL: http://dl.acm.org/citation.cfm?id=358437. 1

[Moo09] MOODY D.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering 35*, 6 (Nov. 2009), 756–779. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5353439, doi:10.1109/TSE.2009.67. 1

[PBG98] PETRE M., BLACKWELL A., GREEN T.: Cognitive questions in software visualization, 1998. 3

[PdQ*06] PETRE M., DE QUINCEY E., ET AL.: A gentle overview of software visualisation. *PPIG News Letter* (2006), 1–10. URL: http://www.ppig.org/sites/default/files/2006-Petre.pdf. 3

[SCGM00] STASKO J., CATRAMBONE R., GUZDIAL M., MCDONALD K.: An evaluation of space-filling information visualizations for depicting hierarchical structures. *International Journal of Human-Computer Studies 53*, 5 (2000), 663–694. URL: http://linkinghub.elsevier.com/retrieve/pii/S1071581900904208, doi:10.1006/ijhc.2000.0420. 2

[Sch11] SCHULZ H.-J.: Treevis. net: A tree visualization reference. *Computer Graphics and Applications, IEEE 31*, 6 (2011), 11–15. 2

[SHS11] SCHULZ H.-J., HADLAK S., SCHUMANN H.: The design space of implicit hierarchy visualization: A survey. *Visualization and Computer Graphics, IEEE Transactions on 17*, 4 (2011), 393–411. 2

[Sta84] STANDISH T. A.: An essay on software reuse. *Software Engineering, IEEE Transactions on*, 5 (1984), 494–497. 1

[Tel14] TELEA A. C.: *Data visualization: principles and practice*. CRC Press, 2014. 1

[War13] WARE C.: *Information visualization: perception for design*, 3rd ed. Interactive technologies. Elsevier, 2013. 2

[Wat05] WATTENBERG M.: A note on space-filling visualizations and space-filling curves. In *Information Visualization, 2005. INFOVIS 2005. IEEE Symposium on* (2005), IEEE, pp. 181–186. 3

[WWDW06] WANG W., WANG H., DAI G., WANG H.: Visualization of large hierarchical data by circle packing. In *Proceedings of the SIGCHI conference on Human Factors in computing systems* (2006), ACM, pp. 517–520. URL: http://dl.acm.org/citation.cfm?id=1124851. 2