2016

On Using Tree Visualisation Techniques to Support Source Code Comprehension

Ivan Bacher
Technological University Dublin, ivan.bacher@tudublin.ie

Brian Mac Namee
University College Dublin, Ireland

John D. Kelleher
Technological University Dublin, john.d.kelleher@tudublin.ie

On using Tree Visualisation Techniques to support Source Code comprehension

Ivan Bacher
Dublin Institute of Technology
Dublin, Ireland
ivan.bacher@dit.ie

Brian Mac Namee
University College Dublin
Dublin, Ireland
brian.macnamee@ucd.ie

John D. Kelleher
Dublin Institute of Technology
Dublin, Ireland
john.d.kelleher@dit.ie

***Abstract*—This paper presents a design study that investigates the use of compact tree visualisations to provide software developers with an overview of the static structure of a source code document within a code editor in order to facilitate source code understanding and navigation. A prototype is presented which utilises an icicle tree visualisation to encode the control structure hierarchy of a source code document, as well as a circular treemap visualisation to encode the scope hierarchy of a source code document. An overview of the prototype and its functionality is given as well as a detailed discussion on the design rationale behind the tool. Possible applications and future work plans are also discussed.**

I. Introduction

Software development is a complex undertaking composed of several activities which include: designing code, writing code, modifying code, and verifying code. Previous studies [1] [2] have shown, that during these high-level activities, software developers dedicate the majority of their time to understanding source code. This includes comprehending the many kinds of relations (e.g. dependency relationships between packages) and many types of hierarchies (e.g. the package-file-class-method-statement hierarchy) within the code. Hence, efficiently understanding source code is an important problem in software development.

Software developers typically use source code editors rather than general purpose text editors for the reading, writing and editing of source code. To facilitate source code understanding, it is important to maximise the readability of source code [3]. Source code editors typically have features, such as syntax highlighting and pretty printing, which increase readability by modifying the typographic appearance of source code. Previous studies [4] [5] have shown that the typographic appearance of source code can influence the speed and accuracy of comprehension, by making the structural and syntactical composition of source code more visible. This can be achieved through the use of indentation, spaces, line-breaks, colour, and variations in font-face. However, Clifton [3] states that the usefulness of these techniques diminishes when parts of control structures are widely separated or heavily nested. The available screen real estate of a source code editor is typically limited and source code documents are frequently too large to be displayed in the available space. 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 created by a source code document and their location within it [6].

This paper presents a design study which investigates the use of a circular tree-map and an icicle tree to encode the hierarchical structure of a source code document. The main contribution of the paper is the utilisation of these tree visualisations techniques in combination with a source code editor to provide software developers with an *overview* of the static structure of a source code document, in order to facilitate source code understanding and navigation. Following Hornbaek and Hertzum [7, p.511], we define 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."* The remainder of the paper is structured as follows. Section II describes various tree visualisation techniques as well as their usefulness. Section III defines the goal of the proposed visualisation approach. Section IV presents the design rational of the proposed approach. Section V proposes applications of the proposed approach. Section VI discusses the most important aspects of the study as well as implications for future work.

II. Related work

For hierarchical data, the parent-child relations of nodes in a hierarchy are an important aspect to be visualised. Thus, many techniques have been developed for the display of hierarchically structured information. Schulz et al. [8] split tree visualisation techniques into two categories: explicit and implicit techniques. Explicit tree visualisation techniques show parent-child relations as straight lines, arcs, or curves. Node-link diagrams (Figure 1) are a commonly used example. Implicit tree visualisation techniques represent parent-child relations by the use of juxtaposition, overlap, or containment. These techniques include tree maps (Figure 2), circular tree-maps (Figure 3) and icicle trees (Figure 4). The goal of this work is to provide software developers with an overview of the various relations and hierarchies within a source code document in the form of a compact visual representation. Thus, we will focus on implicit tree visualisations, as these are typically more space efficient than explicit ones.

Treemaps [9] (Figure 2) provide an overview of an entire hierarchy and are generated by recursively slicing the available

978-1-5090-3850-3/16 $31.00 © 2016 IEEE
DOI 10.1109/VISSOFT.2016.8

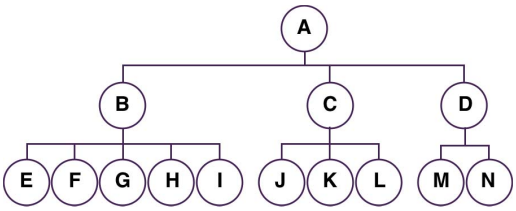

Fig. 1. Node link diagram

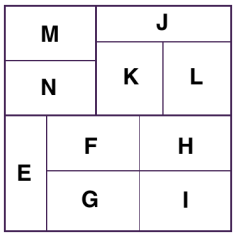

Fig. 2. Tree-map

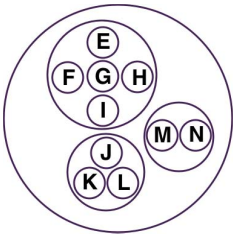

Fig. 3. Cicular tree-map

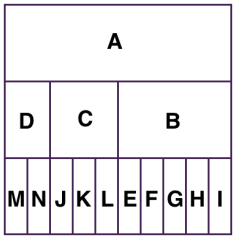

Fig. 4. Icicle tree

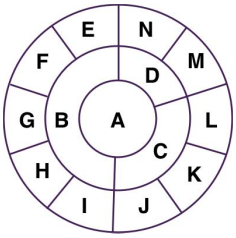

Fig. 5. Sunburst diagram

display space into smaller boxes for each level of the hierarchy. 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 treemaps [10] (Figure 3) are an extension of the treemap approach that use nested circules instead of boxes, which makes it easier to see groupings and hierarchical organisation. Sunburst diagrams [11] (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 (also known as adjacency diagrams) [12] (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 within a hierarchy. Icicle trees are able to adopt either a vertical or horizontal layout, making them highly adaptive to space and layout constraints.

Several experiments have investigated the effectiveness of various tree visualisation techniques. For example, Cawthon and Moere [13] used an online survey of 285 participants to measure the perceived aesthetic as well as the efficiency and effectiveness of retrieval tasks across a set of 11 different tree visualisations techniques. The study establishes a quantitative ranking of the tree visualisation techniques and shows a correlation between latency in task abandonment and erroneous response time in relation to the perceived aesthetics of a visualisation. Stasko et al. [14] performed empirical studies on the usefulness of treemaps and sunburst diagrams for depicting and navigating file hierarchies. Their findings indicate that the sunburst method aided in task performance in terms of correctness and completion time compared to the treemap method. Furthermore, participants indicated a preference for the sunburst method over the treemap method.

In the context of using tree visualisation techniques to depict the hierarchical structure of source code, Bacher et al. [15] evaluated the use of the icicle tree visualisation technique in combination with a source code editor for visualising the hierarchical structure of a source code document. Figure 6 shows a screenshot of this. Nodes within the icicle tree encode the hierarchical structure of the HTML document displayed in the source code editor, and are represented as graphic primitives. Parent-child relations are encoded by horizontal adjacency, meaning that child nodes are placed to the right of their parent. The highlighted node within the icicle tree corresponds to the structural element at which the cursor is located in, within the source code editor. The study showed that for counting tasks, participant accuracy seemed to increase when the visualisation was present. Additionally, completion times were also generally lower for participants using the overview visualisation.

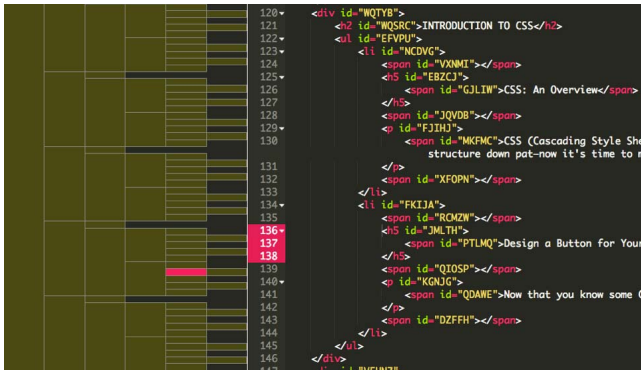

Fig. 6. Prototype interface composed of an overview visualisation and source code editor

The studies described above illustrate that different tree visualisation techniques can be used to depict the hierarchical structure of an information space. For this work, we explore the use of a subset of the described tree visualisation techniques to encode the various types of relations and hierarchies that exist in source code.

III. Dimensions of Software Visualisation

Source code contains many types of relations and hierarchies. It makes sense to consider using tree visualisation techniques to encode some of these to facilitate source code understanding and navigation. In this design study we consider the visualisation of two specific types of hierarchies within source code: scope hierarchy and control structure hierarchy.

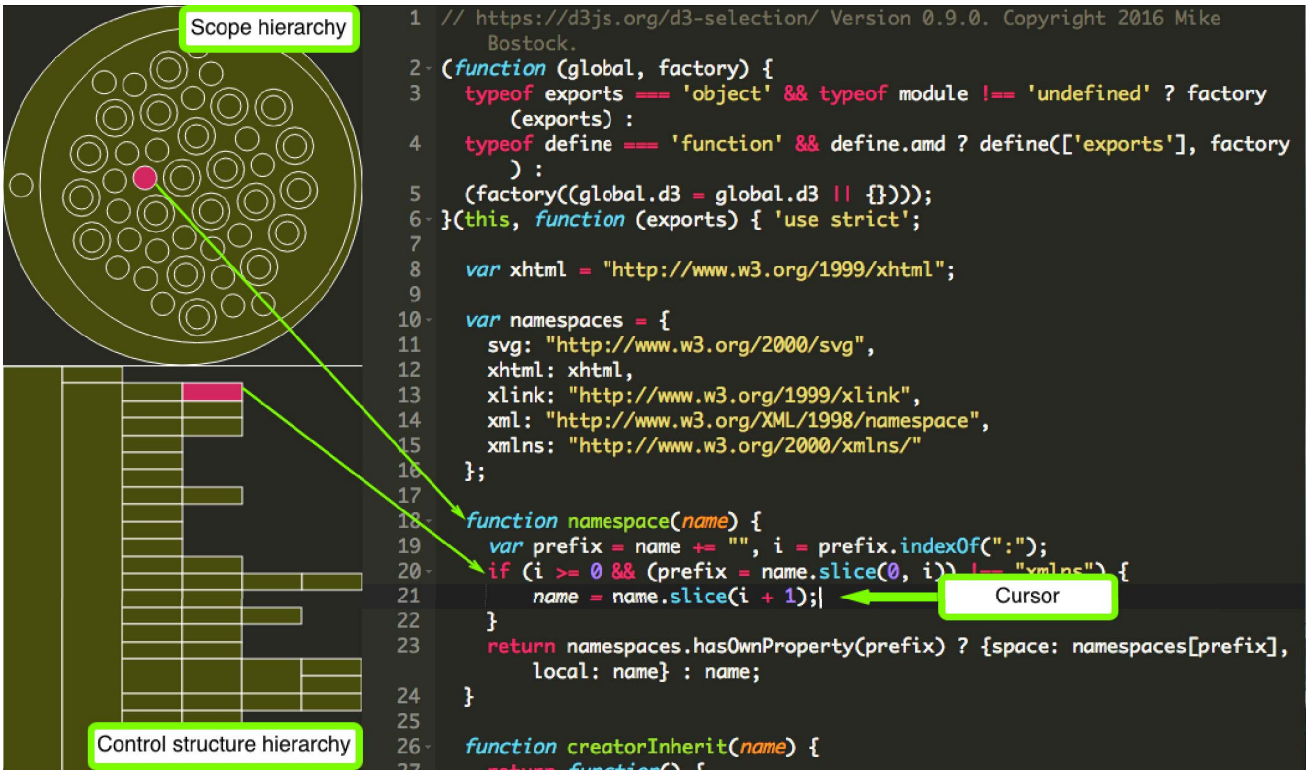

Fig. 7. Version one of prototype interface

Using the five dimensions model of Maletic et al. [16] (task, audience, target, representation, and medium), we define the goal of the proposed visualisation approach.

A. Task - why is the visualisation needed?

In order to comprehend the structure and behaviour of a software system, software developers must read through its source code. This process is also known as tracing [17], which involves scanning through the source code in either a forward or backward direction. Tracing involves both semantic and syntactic knowledge [18]. Semantic knowledge is relatively independent of any particular language and invokes an understanding the basic concepts such as looping structures, and recognising design patterns in code [17]. Syntactic knowledge is language specific and allows semantic structures to be recognised in a particular language [17]. Thus, the readability of source code is an important aspect to consider in the context of comprehension. By making the structural composition of source code more visible (for example though the use of indentation [5] and control structure diagrams [19]) the speed and accuracy of comprehension can be increased [5] [20].

Clifton [3] states that the usefulness of indentation diminishes when parts of control structures are widely separated or heavily nested. This can make it difficult for readers to skip around a group of statements or find the path back from the end to the beginning of a control structure. Additionally, in languages such as JavaScript functions declarations can be nested. With each function creating a new scope, developers may become distorted when navigating though the heavily nested information space. By using tree visualisation techniques to encode the scope and control structure hierarchies within source code, software developers could be helped to gain awareness of the information space and their location within it. Thus, this approach may be able to circumvent the cognitive burden introduced by scrolling and heavy nesting.

B. Audience - who will use the visualization?

The main audience targeted by the presented approach are users that deal with source code. These are typically software developers, however, software testers and software project managers are also included.

C. Target - what is the data source to represent?

In this work we focus on the scope and control structure hierarchy within a source code document. We believe that these hierarchies are important to software developers as the scope hierarchy refers to the set of all entities that are visible or names that are valid within a portion of a program, and the control structure hierarchy determines the control flow of a program to a certain extent.

In programming languages scope controls the visibility and lifetime of variables and parameters. A scope context can nest other scopes or be nested within a parent scope. Each scope context can contain declarations or definitions of identifiers

as well as statements and expressions. Simply put, scope is an enclosing context within which values and expressions are associated. Most languages implement block scope, which represents a list of statements wrapped with curly braces. This means that all variables defined within a block are not visible from the outside and can be released when the execution ceases. Languages such as JavaScript, however, implement function scope. This means that in order to create a new scope, a new function must be implemented instead of control structures such as for, while, if, and switch.

Control structures within source code determine the order in which statements are executed. While control structures may vary in regards to the programming language of choice, basic control structures include if-else and switch statements as well as loop and error handling constructs. These statements and constructs can be nested and span over many lines of code.

D. Representation - how to represent the visualisation?

Figure 7 illustrates our prototype interface, which is composed of an icicle tree, a circular treemap, and a source code editor. Nodes within the icicle tree encode the control structure hierarchy of the source code document located in the source code editor, and are represented as graphical primitives. Parent-child relations are encoded by horizontal adjacency, meaning that child nodes are placed to the right of their parent. Nodes within the circular treemap encode the scope hierarchy of the source code document located in the source code editor, and are represented as nested circles. Parent-child relations are encoded by containment, thus, child nodes are drawn within circles representing parent nodes. Depending on which control structure and scope the text cursor is located in within the source code editor, the corresponding nodes within the icicle tree and circular treemap are highlighted. The layouts of both the icicle tree and curricular treemap are calculated using a space-filling algorithm, in order to use the available display space and avoid scrolling.

E. Medium - where to represent the visualization?

The medium of choice for this work is a standard computer display, which should support a minimal resolution of 1024x768.

IV. Design rational

To facilitate the design and creation of software visualisation tools, several design guidelines and principles have been proposed in the literature [21] [22], ranging from general information visualisation to specific software visualisation guidelines. While all 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, Natural mapping, Awareness, and Interaction.

Language specific: A large variety of programming languages exist, including C, Java, Swift, and JavaScript. These languages typically contain different control structures and a different implementation of a scope hierarchy. Therefore, an open question remains about the usability and effectiveness of the visualisation approach that this work presents, as some languages may be easier to comprehend compared to other languages. For the initial prototype, illustrated in Figure 7, the programming language of choice is JavaScript, as source code written in JavaScript can be heavily nested in terms of control structures and scope hierarchy.

Order adjacency: In the context of control structures, the ordering of these structural elements is particularly important as it typically specifies the flow of control in which individual statements, instructions, or function calls are executed or evaluated. Thus, we believe that 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 top to bottom projection of the control structures, similar to the one in the source code.

Natural mapping: Scope is an enclosing context within which values and expressions are associated. Hence, we believe that a tree visualisation technique which shows parent child relations using containment is best suited for the task of visualising the scope hierarchy within a source code document. These techniques include treemaps and circular treemaps. Treemaps are well known, however, the technique falls short in conveying the global structure of a hierarchy as non-leaf nodes are not shown [21, p. 229]. Additionally, using rectangular shapes makes treemaps difficult to interpret [22, p. 437]. Although circular treemaps remain somewhat experimental and have mainly been used to depict archives and directories of digital files [23], they make the hierarchical structure much clearer than treemaps as they allow empty regions between circles [22, p. 437]. Hence, we believe that the circular treemap technique is well suited for encoding the scope hierarchy of a source code document, as the hierarchical structure is explicitly shown.

Awareness: The goal of both the scope hierarchy and control structure hierarchy visualisations, depicted in Figure 7, is to provide software developers with an overview of the information space, and show their current location within the information space. The icicle tree and circular treemap displayed in Figure 7 encode 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 and circular treemap. Figure 7 illustrates this feature, as the cursor is currently located on line 21 within the source code editor, which contains an if statement in a local scope context. The corresponding nodes are highlighted in the icicle tree and circular treemap.

Interaction: Software developers should be able to use the overview visualisations to navigate though the source code located in the source code editor. By interacting (clicking) with the visualisations, software developers are able to rapidly move to any location within the information space, changing the

point of focus in the source code editor to the corresponding line of code. In the current prototype (Figure 7), users are able to click on a node, within the icicle tree or circular treemap, in order to navigate to the line of code of the corresponding structural element.

V. Application

A common scenario in software development is that a software developer receives the task of re-factoring existing code. This code can be familiar or unfamiliar to the developer and typically requires extensive effort to comprehend.

Before reading the code, the developer can use the prototype visualisations to get an insight into the underlying complexity of the code fragment in terms of scope and control structure nesting. Additionally, once the process of re-factoring the code is complete several versions of the overview visualisation can be compared and used as an aid for explaining the changes made to the corresponding code to team members or managers.

Functions can span over a number of lines of code. This can result in developers having to scroll through a source code document as the available screen real estate of a computer display is typically limited. This can cause disorientation, however, the overview visualisations presented in this paper can be used to limit disorientation by visually showing developers their current location within the control structure and scope hierarchy of the corresponding source code document. Additionally, the overviews visualisations can also be used to navigate the corresponding source code document, allowing developers to navigate to a specific line of code with ease.

VI. Conclusion

This paper has presented a design study which investigates the use of compact tree visualisations to provide software developers with an overview of the static structure of a source code document within a code editor. The main goal of the proposed approach is to facilitate source code navigation and understanding, by allowing developers to use visual representations of the hierarchies contained within source code. However, in order to obtain information regarding the usefulness and usability of the proposed approach, empirical evaluations must be conducted. We consider this to be future work and plan to conduct several evaluations in order to gather quantitative as well as qualitative data relating to the usefulness and usability of the proposed approach. As many other tree visualisation techniques exist, a promising direction for future work also includes the investigation of the appropriateness of these techniques, or a combination of the techniques, for the visualisation of the many types of relations and hierarchies within source code. Additionally, we also plan to investigate the use of colour for encoding control structure or scope types within the visualisations. An initial version of the presented prototype is available at *http://tiny.cc/pgq6by*.

References

[1] T. A. Standish, "An essay on software reuse," *Software Engineering, IEEE Transactions on*, no. 5, pp. 494–497, 1984.

[2] T. A. Corbi, "Program understanding: Challenge for the 1990s," *IBM Systems Journal*, vol. 28, no. 2, pp. 294–306, 1989.

[3] M. H. Clifton, "A technique for making structured programs more readable," *ACM Sigplan Notices*, vol. 13, no. 4, pp. 58–63, 1978.

[4] R. M. Baecker and A. Marcus, *Human factors and typography for more readable programs*. ACM, 1989.

[5] R. J. Miara, J. A. Musselman, J. A. Navarro, and B. Shneiderman, "Program indentation and comprehensibility," *Communications of the ACM*, vol. 26, no. 11, pp. 861–867, 1983.

[6] A. Cockburn, A. Karlson, and B. B. Bederson, "A review of overview+ detail, zooming, and focus+ context interfaces," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, p. 2, 2009.

[7] K. Hornbæk and M. Hertzum, "The notion of overview in information visualization," *International Journal of Human-Computer Studies*, vol. 69, no. 7, pp. 509–525, 2011.

[8] H.-J. Schulz, S. Hadlak, and H. Schumann, "The design space of implicit hierarchy visualization: A survey," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 4, pp. 393–411, 2011.

[9] B. Johnson and B. Shneiderman, "Tree-maps: A space-filling approach to the visualization of hierarchical information structures," in *Visualization, 1991. Visualization'91, Proceedings., IEEE Conference on*. IEEE, 1991, pp. 284–291. [Online]. Available: http://dx.doi.org/10.1109/VISUAL.1991.175815

[10] W. Wang, H. Wang, G. Dai, and H. Wang, "Visualization of large hierarchical data by circle packing," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006, pp. 517–520. [Online]. Available: http://dl.acm.org/citation.cfm?id=1124851

[11] J. Stasko, R. Catrambone, M. Guzdial, and K. Mcdonald, "An evaluation of space-filling information visualizations for depicting hierarchical structures," *International Journal of Human-Computer Studies*, vol. 53, no. 5, pp. 663–694, 2000. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1071581900904208

[12] J. Heer, M. Bostock, and V. Ogievetsky, "A tour through the visualization zoo." *Commun. Acm*, vol. 53, no. 6, pp. 59–67, 2010.

[13] N. Cawthon and A. V. Moere, "The effect of aesthetic on the usability of data visualization," in *Information Visualization, 2007. IV'07. 11th International Conference*. IEEE, 2007, pp. 637–648.

[14] J. Stasko, R. Catrambone, M. Guzdial, and K. McDonald, "An evaluation of space-filling information visualizations for depicting hierarchical structures," *International Journal of Human-Computer Studies*, vol. 53, no. 5, pp. 663–694, 2000.

[15] I. Bacher, B. M. Namee, and J. D. Kelleher, "Using Icicle Trees to Encode the Hierarchical Structure of Source Code," in *EuroVis 2016 - Short Papers*, E. Bertini, N. Elmqvist, and T. Wischgoll, Eds. The Eurographics Association, 2016.

[16] J. I. Maletic, A. Marcus, and M. L. Collard, "A task oriented view of software visualization," in *Visualizing Software for Understanding and Analysis, 2002. Proceedings. First International Workshop on*. IEEE, 2002, pp. 32–40.

[17] S. Cant, D. R. Jeffery, and B. Henderson-Sellers, "A conceptual model of cognitive complexity of elements of the programming process," *Information and Software Technology*, vol. 37, no. 7, pp. 351–362, 1995.

[18] B. Shneiderman and R. Mayer, "Syntactic/semantic interactions in programmer behavior: A model and experimental results," *International Journal of Computer & Information Sciences*, vol. 8, no. 3, pp. 219–238, 1979.

[19] J. H. Cross II, T. D. Hendrix, and S. Maghsoodloo, "The control structure diagram: An overview and initial evaluation," *Empirical Software Engineering*, vol. 3, no. 2, pp. 131–158, 1998.

[20] D. Hendrix, J. H. Cross, S. Maghsoodloo *et al.*, "The effectiveness of control structure diagrams in source code comprehension activities," *Software Engineering, IEEE Transactions on*, vol. 28, no. 5, pp. 463–477, 2002.

[21] C. Ware, *Information visualization: perception for design*, 3rd ed., ser. Interactive technologies. Elsevier, 2013.

[22] W. Huang and P. Eades, *Handbook of human centric visualization*. Springer, 2014.

[23] M. Lima, *The book of trees: visualizing branches of knowledge*. Princeton Architectural Press, 2014.