Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

An Ontology-Based Approach to Visualize Large Software Graphs

Masterarbeit

im Studiengang Informatik

von

Lukas Nagel

Prüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Dr. rer. nat. Jil Ann-Christin Klünder
Betreuer: Dr. rer. nat. Jil Ann-Christin Klünder

Hannover, 24.07.2020

Abstract

New members joining software development teams often have to gather knowledge of a foreign software project. Since such projects can quickly scale up in size, the task of getting an overview of a software code can be overwhelming, when only a folder structure and source code files are available. However, graphs consisting of nodes representing software classes and edges being drawn between nodes that have some form of dependency between them, can aid in these situations, as long as well designed visualizations are available.

In this thesis ontologies are used to impose a clustering on nodes of a graph in an attempt to reduce the cognitive load experienced by users of graph visualization tools. This goal is mainly achieved by a reduction of visual clutter on the screen due to clusternodes being presented instead of the nodes included in them. Additional efforts are made to preserve a user's mental map when expanding a clusternode to visualize its children once more.

Furthermore, intuitive interaction methods with clusternodes and visual aids like a minimap presenting the imposed ontology are developed. Two visualization variations, differing in terms of their display of expanded clusternodes, are also conceptualized. Moreover, possible use cases for the tool are lined out.

All developed aspects are evaluated in a user study consisting of two use case scenarios that participants work through using the tool, before answering questions regarding their experience with it. Results are found to be favourable to both the overall tool visualizing the software graph and the clustering methodology added by this thesis.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Gaining an overview of a foreign software code can be an overwhelming tasks to many developers joining a team of developers. Graphs visualizing software projects as node-link diagrams can support users in such tasks and allow for an exploration that might lead to interesting new insights like high cohesion values for its different modules. Since the sheer size of modern code collections lead to very large dependency graphs, there is a need for efficient and effective visualizations.

This need is also apparent in the amount of work that has been done in the general field of graph visualization. Due to the large amount of possible use cases, any improvement to such visualizations is bound to impact and assist the work of a multitude of scientific fields and their respective researchers.

Examples for use cases of general graph presentations include, but are not limited to, the exploration of a social web like Facebook friends or a network of researchers and their publications. Such a structure might present the user with inherent groupings of authors who have worked together closely in the past or offer insights into key figures in research communities.

The case that will be expanded on in this thesis however, is a graph representing source code of software. The investigated graph structure specifies classes as nodes, and edges between them as any form of dependency like one class calling another, therefore demonstrating a use case that can quickly scale up to huge numbers of nodes and edges. A simple expansion of all existing class nodes into multiple visual entities representing their individual methods can be seen as one of the most basic ways a presentation of code graphs can quickly increase in scale. Software engineers are often interested in unforeseen groupings in their code or seek to find assistance for the exploration of a new software project.

Modern code collections and their visualization have introduced a number of issues to the scientific community, one rather apparent example being the comprehensibility concerns raised by Bikakis and Sellis [8] that stem from the huge number of nodes and edges in the created graphs. Not only does a

large graph by itself challenge a user's comprehensive abilities, the fact that many visual entities on the screen lead to them overlapping one another and a high amount of edge crossings also negatively impact a user's experience with the tool. Additionally, the high volume of forces originating from the layout algorithm and the resulting amount of movement on the screen are another factor to be considered. Therefore, the overall usability of any visualization tool is impacted by both the number of visual entities to be presented and the interaction techniques offered to the user, with the number of visual entities being directly linked to the size of the visualized software. A graph presentation that overloads the user with a cluttered screen or many different colours will not have much use.

Ontologies present one possible way to share knowledge of concepts and data of a specific domain when seen from the perspective of the research field *knowledge engineering*. When taking the computer science perspective, ontologies present a way to structure a given set of data points in a more comprehensible manner.

The presented thesis offers an approach to code graph visualization utilizing ontologies in order to find a novel solution to the aforementioned challenges, mostly focussing on the aspect of cognitive load. While the usage of ontologies in combination with graphs in prior work has mostly been restricted to the presentation of ontologies themselves as node-link diagrams, the demonstrated approach revolves around an ontology resulting in a hierarchy of clusters created on the graph nodes. These clusters are then used to present a less cluttered visualization that still offers insights into the overall topology of the software, thereby improving its usability. The given concepts can also be transferred to general graphs in order to improve their imagery.

1.2 Research Objective

The overarching goal of this thesis is to develop a way of using ontologies to visualize a graph structure in a way that is comprehensible and visually pleasing while potentially also lowering the cognitive load required to use the tool. An evaluation of the proposed concepts for the use of ontologies and the corresponding visualization by means of a user study is also described and the results are lined out.

This work's main contributions lie in the reduction of a user's cognitive load. In order to achieve this improvement a focus is put on the preservation of the mental map of the subject. The implemented concept is designed to present the dependencies and hierarchies of structures of computer code as a graph. Nodes and edges are, respectively, defined as the graphical entities corresponding to code entities like packages or classes and their dependencies. The tool aims to offer insights into the overarching structure of the software project and other potentially important pieces of information, that are not immediately apparent to a developer who otherwise has only a folder structure, the underlying classes and their respective computer codes to go on.

In order to accomplish the goals in terms of performance gains concerning the cognitive load, this thesis utilizes ontologies by means of hierarchies. They are created by asking experts of the respective software projects questions on inherent groupings that exist within the code while trying especially hard to find combinations that are not immediately apparent in the projects package structure. The groups found are then clustered and represented as large cluster nodes in order to reduce the clutter on the screen and give an overview of the software project.

To ensure the applicability and usefulness of the approach a study is conducted involving ten participants. The evaluation focusses on the tool's performance in terms of visualization comprehensibility as well as the cognitive load required from the user to perform the tasks.

1.3 Structure

The rest of this thesis is structured as follows. Chapter 2 provides the fundamental information necessary to understand the meaning and value of the concepts developed. Most importantly the basics of graphs and ontologies are laid out. Chapter 3 gives an overview of other related work while chapter 4 concerns the concepts for both the use of clusterings and the visual properties of the presentation. In chapter 5 the tool this thesis' concepts are added upon and their implementation are described. Chapter 6 provides information on the study evaluating the developed concepts, with its results being outlined in chapter 7. Following this, chapter 8 discusses the results as well as their limitations and expandability. Lastly, chapter 9 provides a summary of this thesis and provides a number of ideas for future work.

2

Fundamentals

The concepts presented in this work require fundamental knowledge in the field of graph theory. These fundamentals are presented in the following chapter.

2.1 Graphs

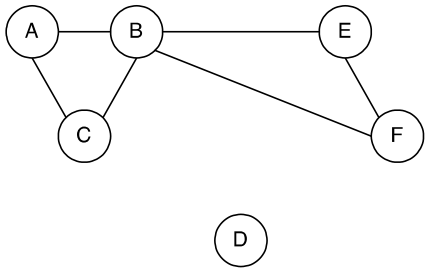
For the purposes of this thesis a graph G is defined as a pair (V, E) where the sets

$$V = \{v_1, v_2, ...\} \tag{2.1}$$

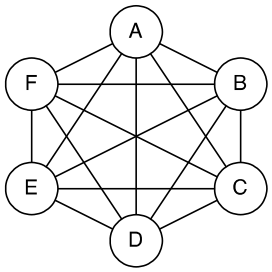
$$E = \{e_1, e_2, ...\} \subset V \times V \tag{2.2}$$

represent the sets of vertices and edges respectively according to Euler [14]. Each edge is defined by a pair of vertices indicating its endpoints, while the number of edges attached to a vertex define its degree. Graphs can be visualized as node-link diagrams where the graph's vertices are depicted by nodes and edges are shown as links between them.

In Software Engineering, more specifically in the case the presented thesis revolves around, node-link diagrams are used to visualize large software projects in an attempt to gather an overview of the underlying dependency structure. For this purpose software classes are presented as nodes. Edges are created amid two nodes if a dependency is evident between the pair of classes. Such dependencies include one class calling a method of another or creating an object defined by the other class. Such software graphs can be used to gain an overview of any code project or to focus on areas of interest once they have been identified. One major issue, however, lies in the number of nodes and links that large software projects produce when visualized in the described manner, since they cause a lot of clutter on the screen. Note that in this case the term "software graph" is not connected with *UML* diagrams.



(a) A basic graph

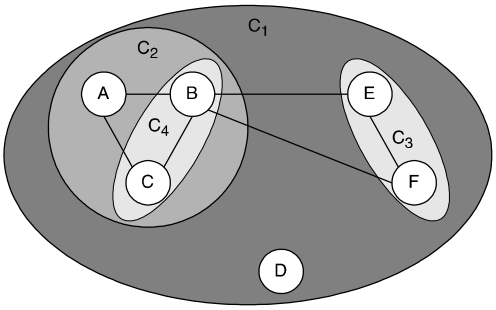


(b) A complete graph of six nodes, also referred to as K_6

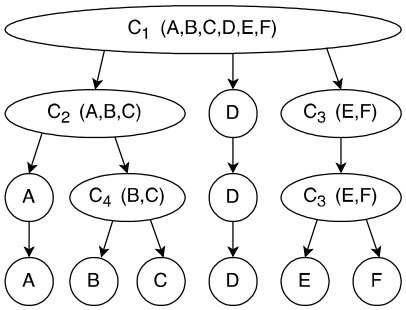
Figure 2.1: Examples for graphs visualized as node-link diagrams.

The field of graph theory has defined and examined a multitude of graph types. Some examples are depicted in figure 2.1 with figure 2.1a showing a simple example of a graph. A complete graph, as shown in figure 2.1b, is defined by Knuth [26] as a graph in which all nodes are connected via direct edges to every other node in the graph. As there exists only one possible complete graph for each number of nodes they are usually presented using the symbol K_n where n is the number of nodes. Therefore, figure 2.1b displays one possible presentation of K_6. If a complete graph only exists locally within a larger graph, it is called a *clique*.

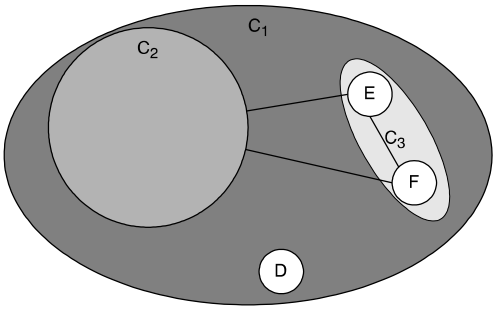
Especially relevant to this thesis is the graph type *compound graph*. A compound graph as described in Kratochvíl [27] is a structure in which nodes can be unified in groups. In an example like the one depicted in figure 2.2 such groupings can be visualized by oval shapes encapsulating the nodes of the graph. Any graph can have a number of different groupings imposed on it. When shown at the same time these different groups can be presented using various colours in an attempt to differentiate them.



(a) A compound graph



(b) The tree of clusters corresponding to the compound graph



(c) The compound graph with cluster C_2 collapsed.

Figure 2.2: Depictions of a compound graph.

2.1.1 Visual Information Seeking Mantra

Shneiderman et al. [38] coined the term *Visual Information Seeking Mantra* as "Overview first, zoom and filter, then details on demand" [38, p. 2]. The mantra is one of the central ideas in the field of large graph visualization and is therefore highly relevant for tools presenting large software graphs. It recommends visualization tool designers to provide their users with an overview of the presented data first. One way to implement this overview is by simply visualizing the initial graph on a blank and "zoomed out" screen, where all the nodes and first iterations of the layout algorithm can be shown at the same time. Secondly, the mantra asks designers to provide users with appropriate interaction techniques, most importantly means to zoom into the presentation and to filter the presented nodes. Basic zoom and pan interactions offer one possible way to realize this part of the mantra. Lastly, "details on demand" is to be interpreted in a way that users should see the details of specific nodes if, and only if, they desire to do so. Such desire can for example be inferred when a user filters for a small set of nodes or zooms in far enough to greatly reduce the number of visualized nodes.

2.1.2 Visual Representation of Graphs

In the presented thesis entities of software, also referred to as code entities, are visualized by visual objects in a node-link diagram. Code entities are displayed as nodes who in turn can be drawn as various shapes like circles, squares or hexagons. Different shapes can be utilized to portray a number of attributes, an example being whether the code entity is a class, a package or a method. Nodes can also exhibit their respective attributes in the form of their colour or a differently coloured border. Additional aspects like size or texture can also visualize more data.

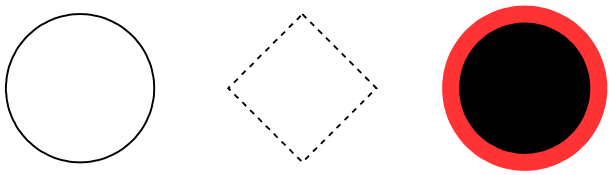


Figure 2.3: Three depictions of nodes: one basic, one with a different shape and outline and one with a border and different colour.

Nodes are connected with edges which depict the dependencies between them. The tool models edges as undirected ones, meaning they have two endpoints instead of one starting- and one endpoint. They are drawn as simple lines due to the desire to reduce the added clutter resulting from the addition of arrowheads. However, edges could still be utilized to display more information than just the existence of any dependency by changing visual aspects like the outline of edges, their colour, thickness or transparency.

Vital to the comprehensibility of a graph is the layout algorithm applied. In recent literature the *force-directed* algorithm has been identified as one of the most optimal options, as discussed in Landesberger et al. [28]. Force-directed layouting attempts to create an aesthetically pleasing visualization by modelling edges with attractive forces similar to springs complying with Hooke's law as described by Rychlewski [37]. Furthermore, repulsive forces modelled after the ones of electrically charged particles governed by Coulomb's law, as mentioned in Amis et al. [5], push nodes away from each other. This leads to unconnected nodes being pushed apart and edges pulling their end points together. Due to their spring-like model edges in force-directed layouts tend to be of uniform length. The algorithm applies the forces iteratively, meaning that the user can be presented with an initial graph layout immediately, while each iteration will adjust node and edge positions in an attempt to improve the visualization aesthetically. Figure 2.4 shows a number of frames from a graph consisting of twelve nodes and twelve edges being laid out via force-directed layouting. It is immediately apparent that the visualization's comprehensibility increases significantly with each iteration step.

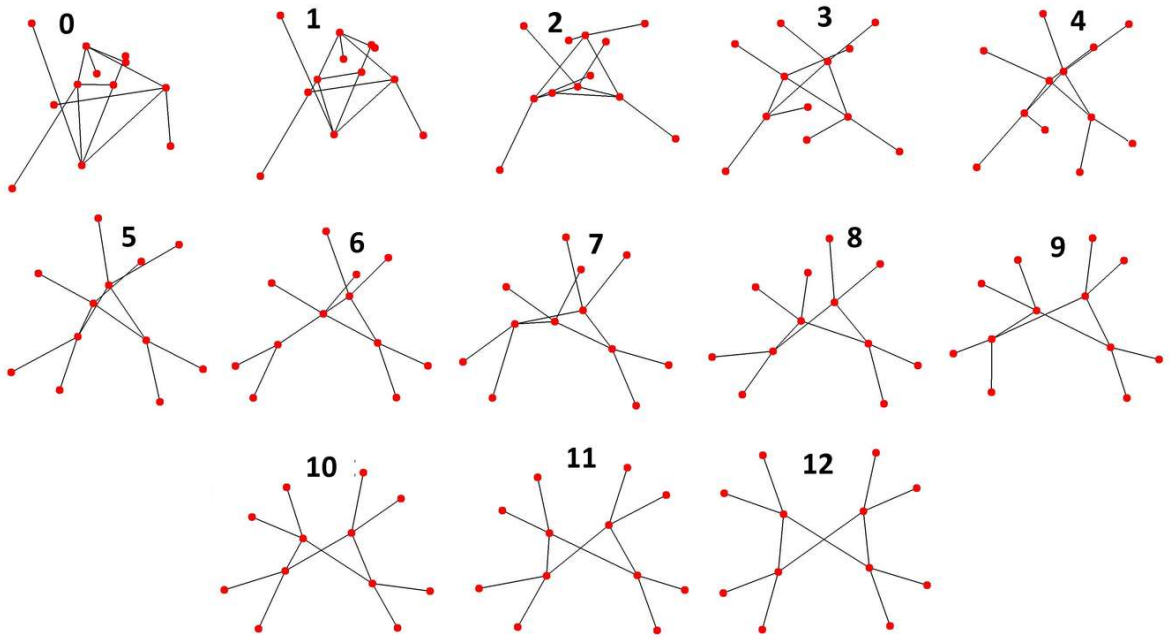


Figure 2.4: Twelve iterations of a force directed algorithm as depicted in Toosi et al. [40]

2.1.3 Clustering

One of the central ways the presented work attempts to reduce the visual clutter on screen, is by grouping sets of nodes as clusters and creating large clusternodes who replace the visual representations of nodes belonging to their cluster. By allowing clusters to consist of both regular and cluster nodes it is possible to create a hierarchy of clusters.

Therefore, this thesis makes use of compound graphs, more specifically their subset of clustered graphs. A compound graph is defined in Kratochvíl [27] as a graph $C = (G, T)$ that consists of a directed or undirected graph $G = (V_G, E_G)$ and a tree $T = (V_T, E_T, r)$ with a root node r and

$$V_G = V_T \tag{2.3}$$

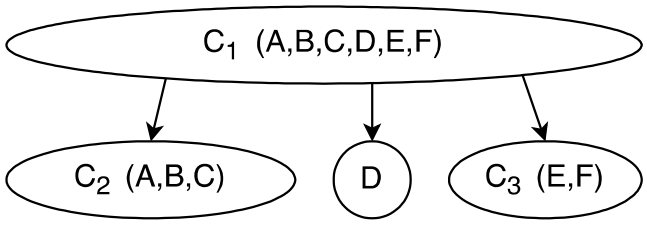
$$a \notin \text{Path}_T(r, b),\ b \notin \text{Path}_T(r, a) \quad \forall\, (a, b) \in E_G. \tag{2.4}$$

Where $\text{Path}_T(r, a)$ is the path from the root node r to the node a in the tree T.

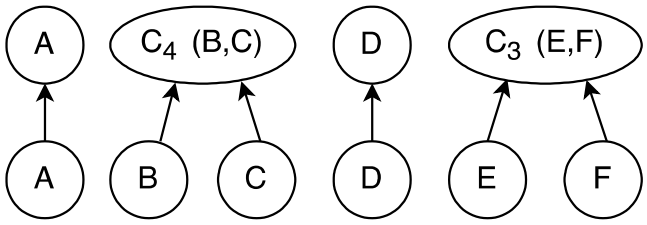
Clustered graphs are a special form of compound graphs, where edges in the graph E_G only exist between nodes that are leaf nodes of the tree, with leaf nodes defined as nodes that do not have any children in the tree T.

The manner in which these clusters are created can be described in two ways that have been named *top-down* and *bottom-up* respectively. Top-down approaches begin with the entire graph as one

large cluster and divide the nodes into multiple smaller ones. After this first iteration, each cluster can once again be separated into any number of clusters less than or equal to the number of nodes in the cluster. Since this process can be repeated until every single node in the graph is represented by its own cluster, a hierarchy can be created this way. Bottom-Up approaches begin with a creation of the lowest level of clusters. This initial set of clusters can then be merged into larger clusters, until all of them are combined into one single cluster representing the initial visualization of the graph at its highest hierarchical level. Examples for both clustering approaches are shown in figure 2.5. Ultimately, the clustering can be seen as a tree of clusters whose leaves are the basic nodes representing code entities. Note that the quality of the clustering greatly influences the quality and expressiveness of the visualization.



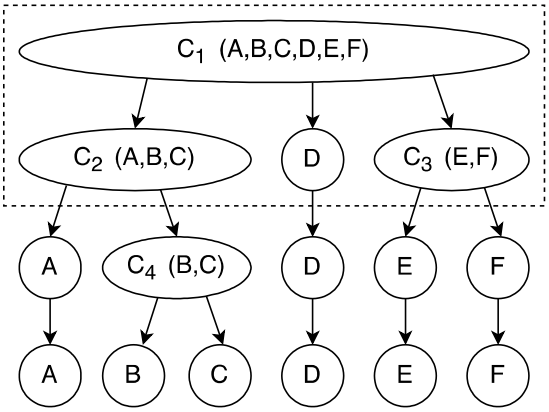
(a) Top-Down Clustering



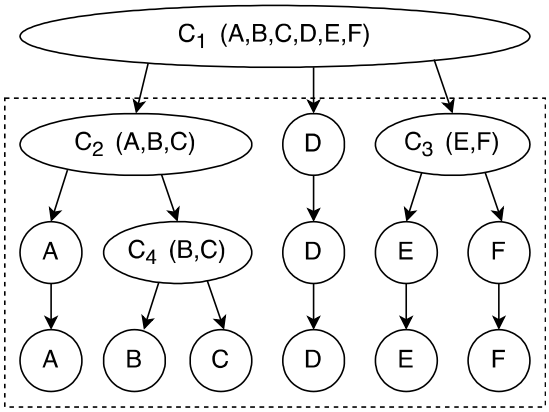
(b) Bottom-Up Clustering

Figure 2.5: Two clustering approaches as depicted by their first clustering iteration. The cluster tree is the one also shown in figure 2.2b.

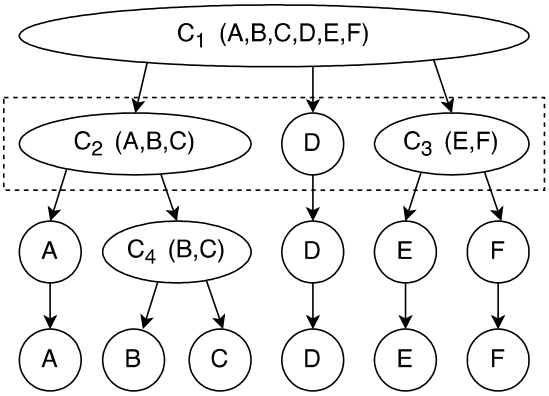
Once a hierarchy is created, any user of the proposed tool is first presented its highest level. Appropriate interaction techniques are required to enable expansion or collapse operations for clusters. This "traversal" of the cluster tree can be designed in five different traversal types coined by Elmqvist and Fekete [12]. Above traversal is defined as showing all nodes included in the current level of abstraction and those above. Details are therefore omitted and a larger focus is put on the aggregates in the hierarchy. The direct opposite, below traversal, is defined in a way that "all nodes below (and including) the current height are rendered" [12, p. 441] and informs the tool's user on how the nodes are clustered. While level traversal only renders nodes at the current level, range traversal extends the rendering to multiple hierarchy levels defined in an interval. Lastly, they also introduce the concept of unbalanced traversals where the user does not choose an entire level of a hierarchy but rather expands and collapses nodes to their liking. Note that nodes in this context are not limited to the nodes representing code entities in the base graph, but also include clusternodes.



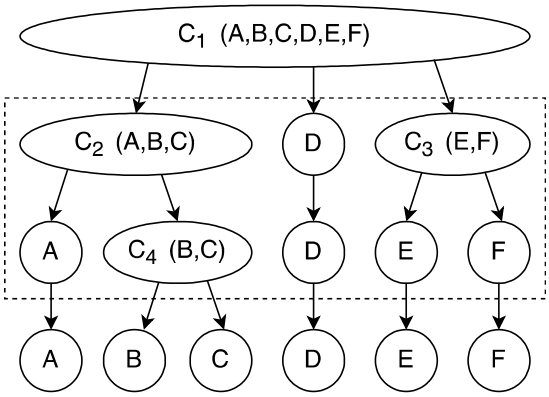
(a) Above Traversal for the second level



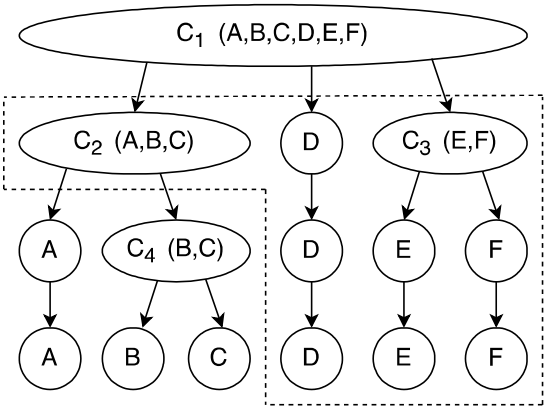
(b) Below Traversal for the second level



(c) Level Traversal for the second level



(d) Range Traversal for the second and third level



(e) Unbalanced Traversal

Figure 2.6: Types of traversals as coined by Elmqvist and Fekete [12]. The dotted shapes indicate the currently rendered nodes. Note that the Unbalanced Traversal depicts the traversal shown in figure 2.2c.

2.2 Ontologies

An ontology in computer and information science is a representation of attributes and relations of concepts or data in a specific domain, as is outlined in Corcho et al. [9]. Ontologies are also used to define these concepts and categories. One of their main use cases is the exchange of information between various different computer programs and services. Generally speaking, ontologies can consist of various different components, the most common of which are the components of "individuals", "classes", "attributes" and "relations". They are lined out in table 2.1.

Individuals	objects
Classes	sets, collections, software classes or types of individuals
Attributes	properties assigned to classes
Relations	connections of classes and individuals

Table 2.1: Common components of ontologies.

It is due to this openness of the ontology, that the concept of an ontology being used to hold the information of a domain is applicable in many fields of scientific research. They are simply used to represent knowledge on any topic. Note that software graphs as described earlier in this chapter can also be seen as ontologies that only consist of classes and relations, while a hierarchy is created using individuals, attributes and relations, when seen from the viewpoint of ontologies.

In many cases two ontologies are created, one of which describes and represents the overall concepts and relations between classes, while the second fills the concepts described in the first with instances. It thereby abides by the "rules" set up by the initial ontology.

In the field of information science ontologies have been used to convey knowledge on a subject area. Multiple publications in the field have attempted to visualize ontologies using node-link diagrams, where entities are modelled as nodes and their relations are represented as links between these nodes. Since software graphs as used in this thesis also convey information on entities like code classes and their relations between one another, not only the hierarchy built upon the graph can be seen as an ontology but so can the base graph itself.

Ontologies are usually defined using one of the various available formal languages, for example the *Resource Description Framework Schema* (RDFS) [2] and the *Web Ontology Language* (OWL) [1], both of which were developed by the *World Wide Web Consortium* (W3C) with the use case of the semantic web in mind.

Figure 2.7 presents an ontology visualized as a node-link diagram using the *Visual Notation for OWL Ontologies* developed in Lohmann et al. [32].

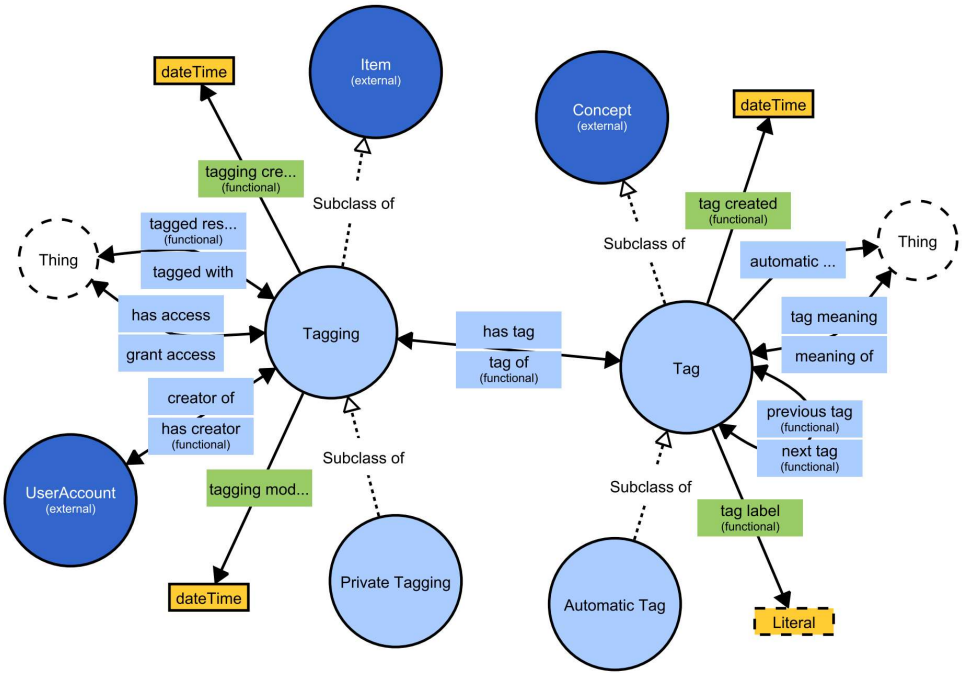


Figure 2.7: An example for an ontology taken from MUTO [30] and visualized using VOWL [32].

Hierarchies as mentioned in section 2.1.3 on clustering are just one example for a collection of concepts where clusters have relations to the encapsulated nodes within them. Their attributes could for example be the number of direct children, meaning that only the next level of the hierarchy is counted, or the total encapsulated leaf nodes. Both metrics are visualized in figure 2.8.

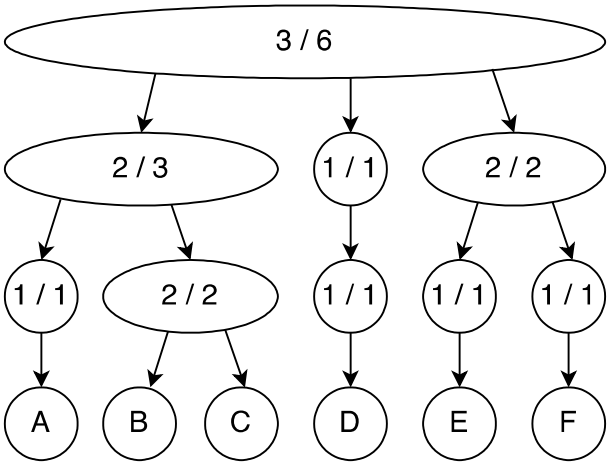


Figure 2.8: The *number of children metric* described in the preceding paragraph visualized. The first number in each node refers to the number of direct children, the second to the total number of leaf nodes.

2.3 Evaluation

2.3.1 Mental Map

As described in Nesbitt and Friedrich [33] as part of proceedings of the International Conference on Information Visualization of 2002 [3], a *mental map* of any user is their personal model of a visualization seen on a screen. For the present example of graph visualization, the mental map is a representation of the displayed graph that can be more or less accurate based on the amount of information provided and the visualization quality. A graph consisting of hundreds of thousands of nodes and edges can not be remembered entirely accurately in its most detailed presentation, but a user might be able to keep a rough outline in mind, provided it has previously been displayed on the screen. Similarly, a part of the graph that has not been presented cannot be assumed as an accurate part of a user's mental map. Generally speaking the map is a concept revolving around any knowledge a user has gathered of a graph's topology.

The most important aspect of a mental map for designers of graph visualization tools is the need for its preservation. When nodes and edges move around significantly after a user has gathered initial information of the graph, such previously collected knowledge is rendered useless and the user has to employ significant effort in order to regain knowledge of the changed graph parts.

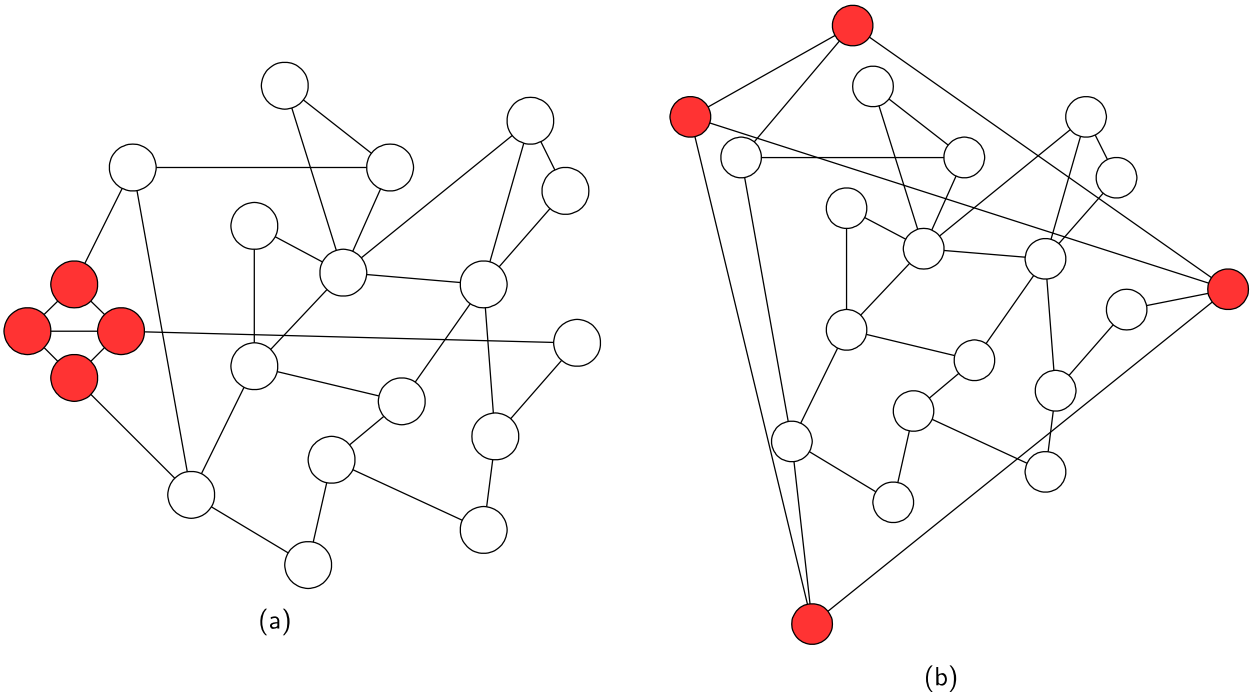


Figure 2.9: A disruption of the mental map. The changed placement of the nodes indicated in red between (a) and (b) majorly reduces the value of any knowledge gathered of the graph's topology on the basis of (a).

Preserving a mental map usually means a preservation of a graphs topology. Archambault et al. [7] define two key principles for mental map preservation in hierarchical graph exploration as "edge conservation" and "connectivity conservation". Edge conservation is coined as "An edge exists between two metanodes m_1 and m_2 if and only if there exists an edge between two leaves in the input graph l_1 and l_2 such that l_1 is a descendant of m_1, and l_2 is a descendant of m_2" [7, p. 901]. On the other hand connectivity conservation is defined as "any subgraph contained inside a metanode must be connected" [7, p. 901].

2.3.2 Cognitive Load

Cognitive load or *memory demand*, according to Haung et al. [22], measures the amount of cognitive resources required to perform a task. For graph visualization such cognitive resources are impacted by the complexity of both the task and the visualization itself. Any reduction in visual complexity means that more mental resources are available to the execution of the given assignment. A major argument for the importance of cognitive load in the evaluation and therefore the design of visualization systems, is that when exploring a large number of nodes and edges, several hypothesis concerning the task completion must be kept in short-term memory at the same time, while also requiring a user to keep the aforementioned mental map in mind.

One of the ways cognitive load has been evaluated for tools similar to the one worked on in this thesis is through self reporting, specifically the *Paas scale* [34], a scale ranging from "very very low mental effort" like riding a bike to "very very high mental effort" such as the effort required to write an exam. The scale encompasses nine different levels and can easily be integrated into a study's survey.

3

Related Work

3.1 Graphs in Software Engineering

Graphs are a frequently occurring data structure in various fields of scientific work and computer science especially. Gansner and North [18] name multiple uses of graphs, including the aspects of specification of both structure and semantics of systems, graphical views of source code structure or relations, and a description of processes composing the creation of new software. They also mention the data structure as being the basis of models used in object-oriented programming paradigms.

Horwitz and Reps [21] present their work on program and system dependence graphs, where the latter is merely an extension of the former. Program dependence graphs visualize single instructions as bubbles and draw arrows between bubbles, when one instruction uses a variable declared in another. While program dependence graphs only allow for the presentation of a singular procedure, the system counterpart includes the functionality to present multiple procedures at the same time. The graph structure worked on in this thesis can be seen as a more modern approach to such system dependence graphs, where only dependencies of entire classes or functions are visualized, instead of a presentation of visual entities for each instruction in a function of the class.

In Joblin et al. [23] graphs are created using nodes to represent developers while drawing edges between developers in some form of relationship, thereby modelling the organizational structure between them. These graphs are then used to examine developer's roles using network-based core-peripheral operationalizations, that they find to align more closely to developer perception than count-based operationalizations. Furthermore, Kiesling et al. [24] make use of graphs in the context of a FLOW analysis. The created FLOW diagrams of information flows between employees of a company are then transformed to networks and used to calculate various metrics revolving around network centrality. Such metrics support the analysis of the original diagrams and can point out inconsistencies within them.

Additionally Van Antwerp and Madey [6] discuss social networks in the context of open source software development. For their network developers and projects are represented as nodes, with edges drawn between a developer and all nodes symbolizing projects that the developer works on. They conclude that developers who have worked together on successful projects likely work together well. It is therefore more likely that they will work together on later projects as well, thereby presenting a use of social networks for open source software development.

3.2 Visualization of Graphs

The importance of research into graph visualization can easily be recognized when considering the amount of research papers published. Google Scholar reports more than 1.5 million search results when querying for "Graph Visualization". Numerous tools have been developed and are still being worked on, which in turn means that there is a multitude of interesting interaction techniques, considered issues and even entire subfields to take into account when working on visualizations of node-link diagrams.

One of the central publications in the field is one by Shneiderman et al. [38] in which the term Visual Information Seeking Mantra is defined as "Overview first, zoom and filter, then details on demand" [38, p. 2]. This mantra is still majorly relevant to many research papers regarding graph visualization. Another fundamental work has been published by Reingold et al. [16]. They presented a force-directed algorithm that has proven to be the most effective layout when aiming for an aesthetically pleasing presentation.

3.2.1 Hierarchical Graph Visualization

Since the focus of the presented thesis lies on the visualization of graphs in a hierarchical manner, a similar focus is applied in the evaluation of related work. A creation of hierarchies and clusters on the nodes in a graph is used in a multitude of visualization tools. This abundance of research efforts into hierarchical graph presentations has brought various issues and corresponding solutions to the attention of developers. One example for such an issue is the need for a preservation of a graph's topology when expanding or collapsing aggregated metanodes. The concept of a mental map of a graph and its preservation has been a major design consideration in the work of various researchers such as the one by Frishman et al. [15].

Attempts to find solutions to these issues have often led to the creation of new interaction techniques like the "merge-at-cut" operation presented by Archambault et al. [7] which they describe as "simply a metanode Merge operation applied to the contents of each open metanode separately" [7, p. 907]. Elmqvist and Fekete [12] provide definitions of multiple traversal types for the exploration of hierarchical graphs, namely "above", "below", "level" and "range" traversal. Another publication by

Landesberger et al. [28] defines the concept of compound graphs as consisting of node-link diagrams at the lowest hierarchy level and enclosures like bubbles around a number of nodes as a visual representation of higher levels. In their work such graphs are created by successive aggregation in a bottom-up approach.

3.2.2 Edge Bundling

Edge bundling is one of the largest points of research regarding the reduction of screen clutter in large graph visualizations. Multiple different approaches attempt to find an algorithm that produces aesthetically pleasing edge bundles, while also being performant in terms of calculation times. Examples for such approaches are the Skeleton-based edge bundling by Ersoy et al. [13], a force-directed method by Holten et al. [20], and a geometry-based edge clustering by Cui et al. [10] who generate meshes in order to find control points through which the edges are forced to create bundles.

3.3 Visualization of Ontologies

Dudáš et al. [11] define ontologies as "a formal explicit specification of a shared conceptualization" [11, p. 2]. In their survey they present a multitude of ontology visualizations, showcasing their various interaction techniques, as well as their strengths and weaknesses. Issues in ontology visualization, mainly the visual depiction of details, are discussed. The most important section of the paper regarding this thesis, however, is their list of basic interaction techniques for visualization tools, that can directly be applied for tools depicting graphs. The list includes the interactions pan and zoom, and the idea of a minimap. Additionally they cite a publication by Gershon et al. [19] mentioning various benefits of visualizations in general, to make an argument for the importance of visualization techniques in the research field of ontologies.

Jambalaya, a tool presented by Storey et al. [39], attempts to depict an ontology as a graph whose nodes can include further nested nodes. They provide an interaction technique presenting more or less detail depending on the zoom level currently used. This approach is also adapted by Wiens et al. [41] who name it "Semantic Zoom". Their publication offers some more insights on the implementation of the zooming technique by presenting multiple layers, specifically the topological, aggregation and visual appearance layers, who are combined with three levels of detail to provide the user with a *focus + context* interaction. Furthermore, issues like the positioning of nodes that are to be presented to the user, after a metanode is expanded, are mentioned and solutions are proposed. Most notably the usage of the free angular space can prove helpful to the presentation of all kinds of hierarchical graphs.

Another tool for the depiction of ontologies is presented by Lohmann et al. who have published multiple iterations of their software *VOWL* [31, 32]. Their research papers give insights into the evaluation of visualization tools and mention interesting suggestions, like the consideration of users suffering from colour blindness who should still be able to utilize the software effectively.

3.4 Evaluation of Graph Visualization

Proper evaluation methods for graph visualization tools have been a point of major research efforts, since rating a visualization is generally a subjective task. Objective measurements of quality can aid developers of visualization tools in picking interaction techniques that have worked best for previously implemented tools. Perer and Shneiderman [35] have described what they believe to be a typical experiment-design as consisting of "20-60 participants, who are given 10-30 minutes of training, followed by all participants doing the same 2-20 tasks during a 1-3 hour session" [35, p. 268]. They also offer insights into the limitations of such evaluations, namely the issue that visualization tools are often used for extended periods of time, whose length greatly exceed the aforementioned 1-3 hour sessions.

Other publications focussing on experiment designs for the evaluation of visualization tools have introduced the concepts of cognitive load and mental effort. Most notably the work of Huang et al. [22] argues that "when working with [...] two visualizations of the same data (suppose that one of the two is bad and the other is good), it is feasible that the same viewer expends more mental effort to compensate for the increased cognitive load induced by the bad visualization, thereby maintaining the same level of performance as with the good one" [22, p. 140]. They offer multiple techniques to measure mental effort, while concluding that the original Paas scale presented in the paper by Paas et al. [34] is "reliable, non-intrusive and sensitive to small changes in memory demand" [22, p. 141].

Their work on cognitive load is extended by Fu et al. [17] with an approach utilizing eye tracking. For measures of the cognitive workload they mention pupil dilation as well as absolute and relative saccade angles. Further arguments for the usage of eye tracking evaluations are also presented, most notably that "while task time and task success tell us what has happened as a result of using a particular tool, eye tracking results help us understand why certain design elements lead to an increase/decrease in speed and accuracy" [17, p. 3].

As an addition to the research on the evaluation of visualization tools and the presentation of various study designs in the aforementioned publications, the taxonomy by Lee et al. [29] presents a number of tasks that could be useful for an evaluation of any tool depicting node-link diagrams. Most notably their finding that "tasks all seem to be compound tasks made up of Amar et al's primitive tasks applied to the graph objects" [29, p. 1], referencing a work by Amar et al. [4], presents a useful insight into user study task design. The taxonomy provides researchers with both a typing and

general descriptions of such tasks, as well as direct examples for user study exercises. Furthermore, Wohlin et al. [42] provide various suggestions for experiments in software engineering, including an explanation of four different types of threats to the validity of study results.

3.5 Separation of this Thesis

The presented thesis offers a novel approach combining ontologies and a node-link representation of software projects visualizing the dependencies between classes. Ontologies are used to create flexible groupings of nodes in the software graph in an attempt to reduce the cognitive load experienced by a user of the tool, while also following Shneiderman's Visual Information Seeking Mantra [38].

4

Concepts

4.1 Ontologies for Graph Visualization

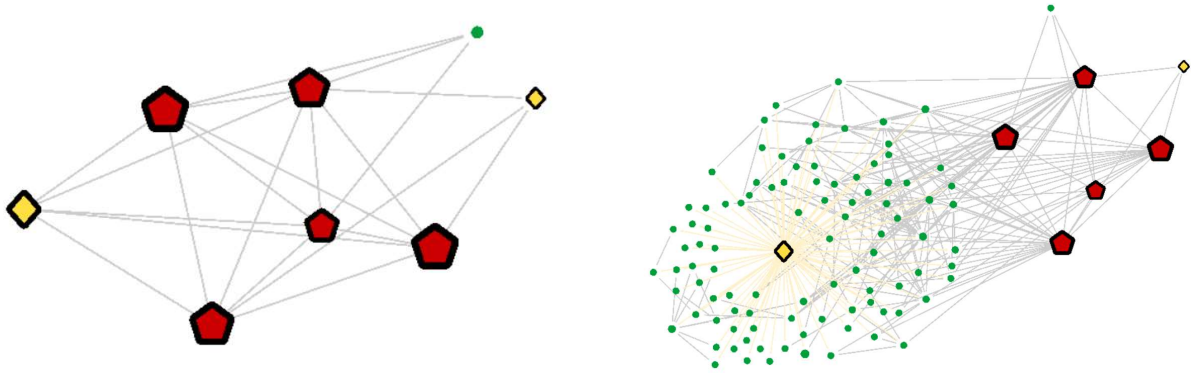
The use of ontologies for the purpose of graph visualization first and foremost facilitates the creation of groupings of graph nodes. More specifically in this thesis, ontologies are utilized as a method of defining such clusters. Various scientific papers have interpreted ontologies as knowledge graphs and a similar theory can be applied to the grouping of graph nodes, as the clustering imposed on the software graph represents another layer of knowledge visualized to aid the user in their task.

Such added knowledge is utilized in this thesis as a means to create groupings on the nodes of a software graph. This method allows for another layer of knowledge to be imposed on the graph and can therefore be presented to the user without completely altering the functionality of the visualization tool. The ontology itself is mostly used as a means to define the clustering in terms of which nodes are grouped together and which groups should be assembled even further into clusters of clusters.

The clustering itself is performed using a bottom-up approach according to the one described in section 2.1.3. Tags assigned to individual nodes in the software graph can be chosen to be grouped in a cluster which will result in a corresponding clusternode appearing in the graph. These tags can be collected in interviews like the one laid out in section 6.1.

A clusternode unites nodes with the chosen tag within itself and inherits all their dependencies, meaning that all dependencies linked to children of the clusternode are instead connected to the parent. Unfolding this clusternode leads to a reappearance of the child nodes in the graph and all links are restored to link to the more specific entities. During unfold operations the clusternode itself loses all edges it had inherited before and is instead linked to its children with a specific parent-child link type. An example can be found in figure 4.1. This ensures that the user can gather the specific knowledge they are looking for, while also being able to distinguish which class nodes were previously hidden in which cluster. The fact that unfolded clusternodes are linked to their children becomes especially useful when multiple clusternodes are unfolded at the same time, or deep hierarchies of

clusternodes are present in the ontology. It should also be noted that clusters in this thesis are fuzzy meaning that there are cases in which a single classnode present in the original software graph can be part of two different clusters, for example a class "teammember" that belongs in two clusters "teamcomponents" and "representations of people". This case is expanded on later in this section.



(a) Before unfolding (b) After unfolding

Figure 4.1: An example for the unfold operation.

These design choices mean that the key principle of edge conservation coined by Archambault et al. [7] is followed for dependency links. It is also important to recognize that the second key principle defined as "any subgraph contained inside a metanode must be connected" [7, p. 901] is not completely adhered to. Nodes that do not have dependencies to the rest of the graph can still be included in a clusternode that is connected to the graph due to dependencies inherited from other children. This design decision is made to allow experts developing ontologies even more freedom, especially since nodes with no dependencies to any other node in the graph present rare edge cases in which a class exists in the project but is not used by any part of the software. Such classes should be evaluated by tool experts and therefore are a meaningful finding of the user's exploration.

Linking the clusternode to its children has the added benefit of impacting the layout algorithm in a way that keeps nodes belonging to the same cluster grouped. Since the force directed algorithm utilized in the tool does not offer a way to assign weights to edges, all links influence their connected nodes with the same force. Ultimately, this leads to the single link between an unfolded clusternode and its children to not be impactful when compared with the combined force of the potentially high number of dependency links. Therefore, nodes are still mostly positioned according to their dependencies while nodes with smaller amounts of dependency links are retained in a position close to other nodes in their cluster. This positioning emphasizes the knowledge visualized in the graph based on the ontology without it overshadowing the main purpose of the visualization visible in the dependency links.

Additionally, functionality allowing a user to assign special tags to clusternodes that are "passed down" to their children is conceptualized. This allows a user to designate a tag once for a parent node that is then added to all children of the parent within the tool, alleviating the work load caused by a user having to assign the tag to each child individually.

It is noteworthy that within clusters some child nodes can be seen as especially important. Usually nodes with higher degrees as defined in chapter 2 are of interest to a user as they have dependencies to a large number of other nodes. An extreme case is a clique that is present within the graph as nodes within the clique are particularly central to the cluster.

For the purposes of this work clusternodes themselves can be distinguished into two different types. Tagnodes are nodes that directly group entities of the original software graph based on their tags. Aggregatenodes can be described as "clusters of clusters", their children can only ever be tagnodes and other aggregatenodes. For the sake of clarity a decision was made not to enable aggregatenodes to directly include class nodes of the original software graph. Instead another tagnode acting as an intermediary is required. This design reduces the amount of variations in results of unfold operations that the user has to expect. Unfolding tagnodes will only show additional base nodes in the presented graph, while an unfold operation of aggregatenodes will only ever yield further tag- or aggregatenodes. All types of nodes and links resulting from these distinctions can be found in table 4.1.

Base node	Node type representing software classes. Is part of the original software graph.
Tagnode	Type of clusternode that only ever groups base nodes.
Aggregatenode	Type of clusternode that only ever groups tagnodes and other aggregatenodes.
Dependency link	Link type connecting two nodes whose corresponding classes depend on one another in some way.
Tagnodelink	Link type connecting an unfolded tagnode and its children.
Aggregatenodelink	Link type connecting an unfolded aggregatenode and its children.

Table 4.1: Types of nodes and links.

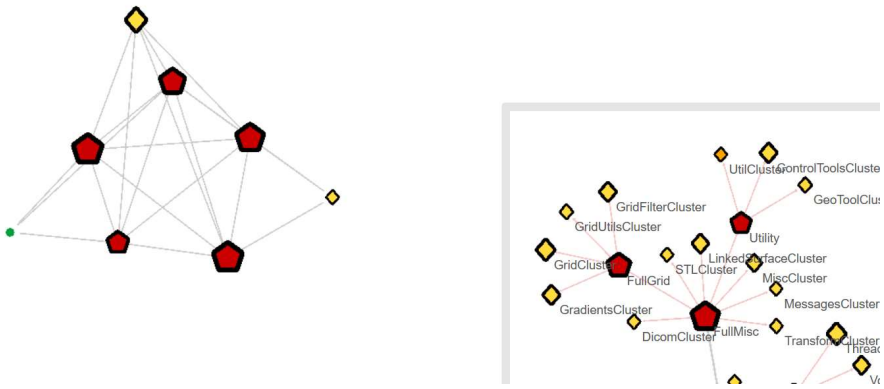
When offering the user a method to unfold clusters, it is easy to imagine the need for a fold operation enabling a quick return to the graph shown prior to the unfolding. A fold operation requires that all children of the folding clusternode need to be removed from the graph, which in turn means a removal of all links connecting the clusternode and its children. Simultaneously, all edges representing dependencies must again be inherited from the children to the parent, meaning that they need to be removed from the disappearing entities, while new dependency edges have to be drawn linking to the clusternode. The high amount of edges that need to appear or be removed in any of these operations can lead to edge cases needing consideration. Most of these edge cases stem from the high variety of potential states of nodes linked by edges representing dependencies. A class node that is shown

in the graph can be linked to another node of the original base graph. However if the other node is part of a cluster whose clusternode is still folded, the class node needs to be connected to that clusternode instead. A single base node can also potentially be part of multiple different clusters. Constellations such as this one can become even more complicated when deeper hierarchies or nodes with multiple distinct parents are included in the ontology. Some of these edge cases are discussed at a later point in this chapter.

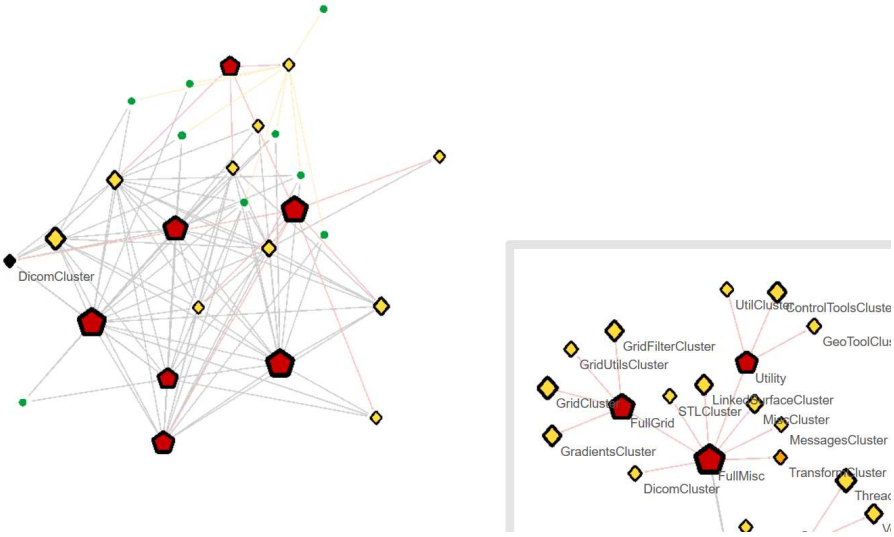
The fold and unfold operations are issued by a double click on the tag- or aggregatenode desired to be folded or unfolded. Double clicks are chosen since users are used to opening files on their computers with the same interaction. With the two options being opposite of one another and every node only having one of the two possible fold states at any given point in time, a decision is made to use the same interaction for both operations. Furthermore, it also became clear over the course of the implementation of the presented concepts, that an additional interaction method is desirable. Due to the constant and sometimes erratic movement of nodes caused by the large number of entities and the resulting large number of forces, it is not always feasible to reliably perform two clicks in quick succession on a moving target. While this issue can be mitigated by simply pausing the layout algorithm, an alternative was still required to enable reliable interaction with a moving graph. Therefore the key "E" on the keyboard was mapped to also fold or unfold a clusternode depending on its current state. Further information on key choices is provided in chapter 5.

In an attempt to further increase the perceived importance of the ontology to the user, and to allow for easier interaction and overview of the imposed ontology itself, a minimap is implemented. It displays a node-link diagram of all nodes that are used for the clustering in the hierarchy. All aggregatenodes are linked to their aggregated components creating a tree in which all leafs are tagnodes and all other nodes are aggregatenodes. They are positioned by the same layout algorithm that governs the nodes in the main graph. The minimap is positioned in the lower right corner and covers less than a quarter of the provided screen space. This position is again chosen with respect to the other elements on screen, like the information box covering part of the left side of the graph canvas, while the size is estimated based on personal preference and user feedback. The minimap is designed to be an integral part of the visualization and meant to draw attention to itself in order to emphasize the importance of the ontology, while still allowing the main graph and its canvas to be the largest and thereby most dominant part of the available screen space. Also note that in this thesis the terms "minimap" and "ontologygraph" are used interchangeably.

Interactions with the minimap offer an additional design space which presents opportunities to provide the user with intuitive ways of improving the experience with the tool. To keep the tool from overloading the user with a surplus of methods, interacting with the minimap works the same way as interacting with the main canvas. This means that all interaction methods described in chapter 2 like pan, zoom and movement of individual nodes work in the exact same way.



(a) Before unfolding



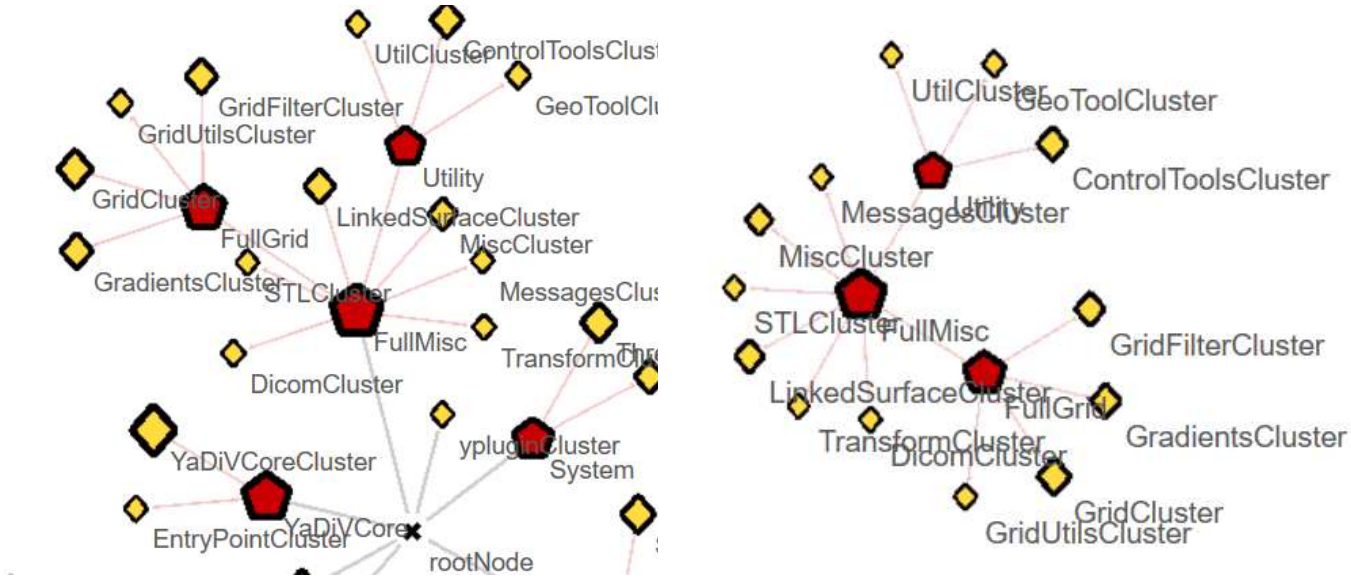
(b) After unfolding

Figure 4.2: An example for the unfold operation using the ontologygraph. The node "utilCluster" is being unfolded.

One interaction that does not fully function in the same way as in the main graph is the folding and unfolding of nodes. The ontology displayed in the minimap corresponds to the ontology imposed on the main graph, meaning that every single node displayed in the minimap is either shown in the main graph or hidden in one of the folded aggregatenodes. It is therefore intuitive that a user can select a node within the minimap to fold or unfold. An interaction method is provided mapping a selection of a node in the ontologygraph in combination with a press of the key "R" on the keyboard onto a fold or unfold operation of the corresponding node in the main graph. To retain some symmetry between fold and unfold operations in the main and ontology graphs the double click interaction was also implemented as a way to initiate a fold operation. For cases in which the selected node in the minimap was not currently visible in the main graph, the decision was made to also unfold all parents

that were folded at the time of the keypress. This means that unfolding a deeply nested node with multiple unfolded parents in the minimap can lead to a large number of other nodes appearing on the screen, even though they might not be of interest to the user. However, this design is still more intuitive than only unfolding parent clusters halfway or completely omitting parent nodes that would have needed to be unfolded to access the node. An example can be found in figure 4.2. Note that the unfolded node in the ontologygraph stays selected when unfolding and a second press of the key "R" executes a fold operation of the just unfolded node. Since the originally unfolded node is now being folded, all parent nodes that were also unfolded to enable a presentation of the chosen node still remain unfolded and in the graph.

The minimap is designed to portray all clusternodes available in the ontology, however, a choice is made to include a single artificial node. This rootnode is implemented as a means to add some more structure to the ontologygraph. While the graph already expanded from the middle of the designated canvas in a radial fashion without the presence of a rootnode, its addition enables an easier overview of the ontology's hierarchy. Now the highest hierarchical node of each branch is the one directly connected to the rootnode and no ambiguity remains as to what node in the branch is the one displayed in the main graph on tool startup. An example can be found in figure 4.3.



(a) With a rootNode

(b) Without a rootNode

Figure 4.3: The minimap once including a rootNode and once without. The second variation does not offer the user an obvious way to find out the hierarchy present in the visualized nodes.

The combination of the minimap and the fold and unfold operations is this thesis' way of conforming with the Visual Information Seeking Mantra coined by Shneiderman et al. [38]. The minimap and the initially presented clustered graph present ways for users to gain an overview of all parts of the visualized software graph, while zooming and filtering can be achieved with simple zoom and pan operations. Details are presented on demand according to the nodes folded and unfolded by the user.

Another interaction method offered to the user is a transformation of node positions in the main graph. When this interaction is triggered by the user the tag- and aggregatenodes currently presented in the main graph are positioned according to their positions in the ontologygraph. Additionally the transformed nodes are pinned, so that their positions do not change with forces of the layout algorithm. Thereby the user also has the chance to adjust the layout of the ontologygraph to their liking by dragging nodes around in it, before transforming the main graph according to the new layout. After the repositioning users who gathered knowledge of the topology of the graph shown in the minimap have an easy time finding the corresponding nodes in the main graph. The most significant drawback of this interaction technique however is the fact that when nodes are transformed to positions according to the ontologygraph, their distances from one another increase substantially. This results in the main graphs edges representing dependencies between the clusternodes becoming long and adding visual clutter. An example for the transformation issued by this technique is presented in figure 4.4.

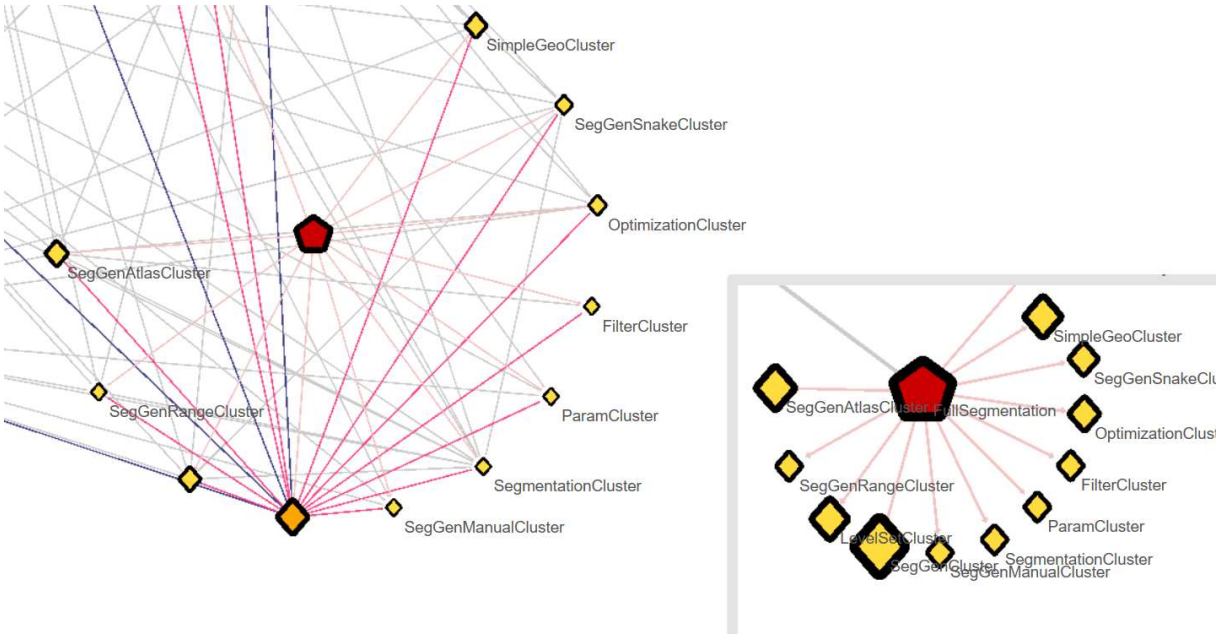
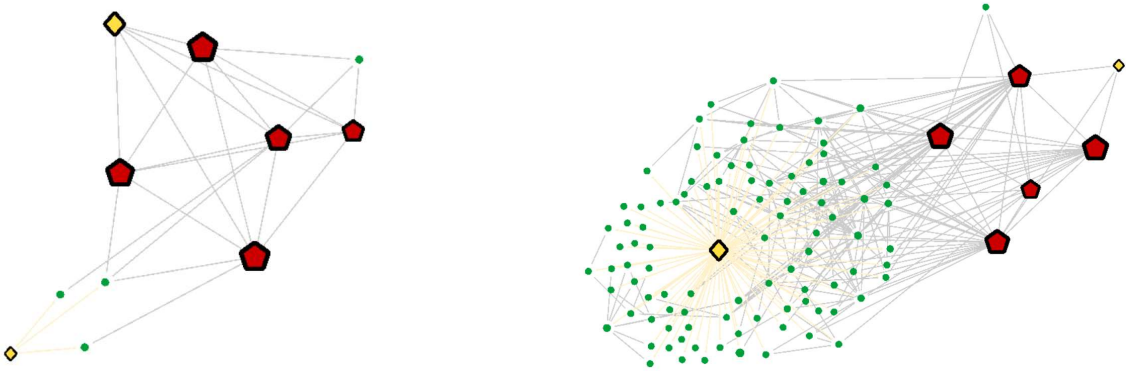


Figure 4.4: Result of the interaction. The same circular structure visible in the structure of the cluster "FullSegmentation" and its children can be found in both the main graph and the ontology graph.

A further important aspect to consider for the design of interactions is the preservation of the mental map. For the purpose of this thesis the term "mental map" is used to describe knowledge the *CodeExplorer's* user has gathered of the topology of the graph presented. In the case of the tool discussed in this work there are two different mental maps to be considered, one of the main graph and one of the ontology graph presented in the minimap. While this adds more cognitive load to the user, the increase is not too high since the ontology graph does not change much after an initial layout. Changes of the nodes in the main graph however, happen frequently and can in some instances be quite significant, for example when a tagnode containing a high number of base nodes is unfolded. The difference of cognitive loads depending on the number of newly appearing nodes on the screen is illustrated in figure 4.5.



(a) An example with a small number of nodes

(b) An example with a large number of nodes

Figure 4.5: An example for the unfold operation causing more cognitive load depending on the number of unfolded nodes.

Any mental map of a test subject is vital to their usage and comprehension of the presented tool, especially when trying to explore an interesting cluster or class. If the position of a node was to drastically change to a different position without any indication of where the desired node might have moved to, the mental map of the user would be disturbed and effectively made useless. Thus rendering any previously gained knowledge of the graph and corresponding interesting insights completely irrelevant, at least for the time it takes the user to find the respective structures in the new graph. Such issues could lead to a largely increased cognitive load and would prevent the tool from being adopted for any kind of research operation.

In an attempt to reduce the cognitive load caused by a disturbance of a mental map a user experiences during unfold operations, children of unfolding nodes are assigned an initial position at which they appear in the graph upon unfolding. The most intuitive choice is to set these positions to the current positions of their parent. This allows for the position of the children to be influenced by the combination of all dependencies assigned to their cluster, as the force directed algorithm had previously positioned the clusternode according to its links. Therefore, all children are starting out in

a position that is already a better fit than simple random coordinates would have been, which results in less required movement. By setting an initial position the tool can rely on the layout algorithm to find suitable locations for nodes while also adding a "blooming" effect as shown in figure 4.6. The "blooming" enables a user to trivially remember which clusternode the children originated from or to quickly recognize which nodes have just been added to the graph in comparison to which nodes had already been visible before the operation. In order to keep confusion to a minimum it is decided that there would not be a difference in mental map preservation methodology between the different node types. All unfold operations set the initial position of the appearing children to the current position of the unfolding clusternode.

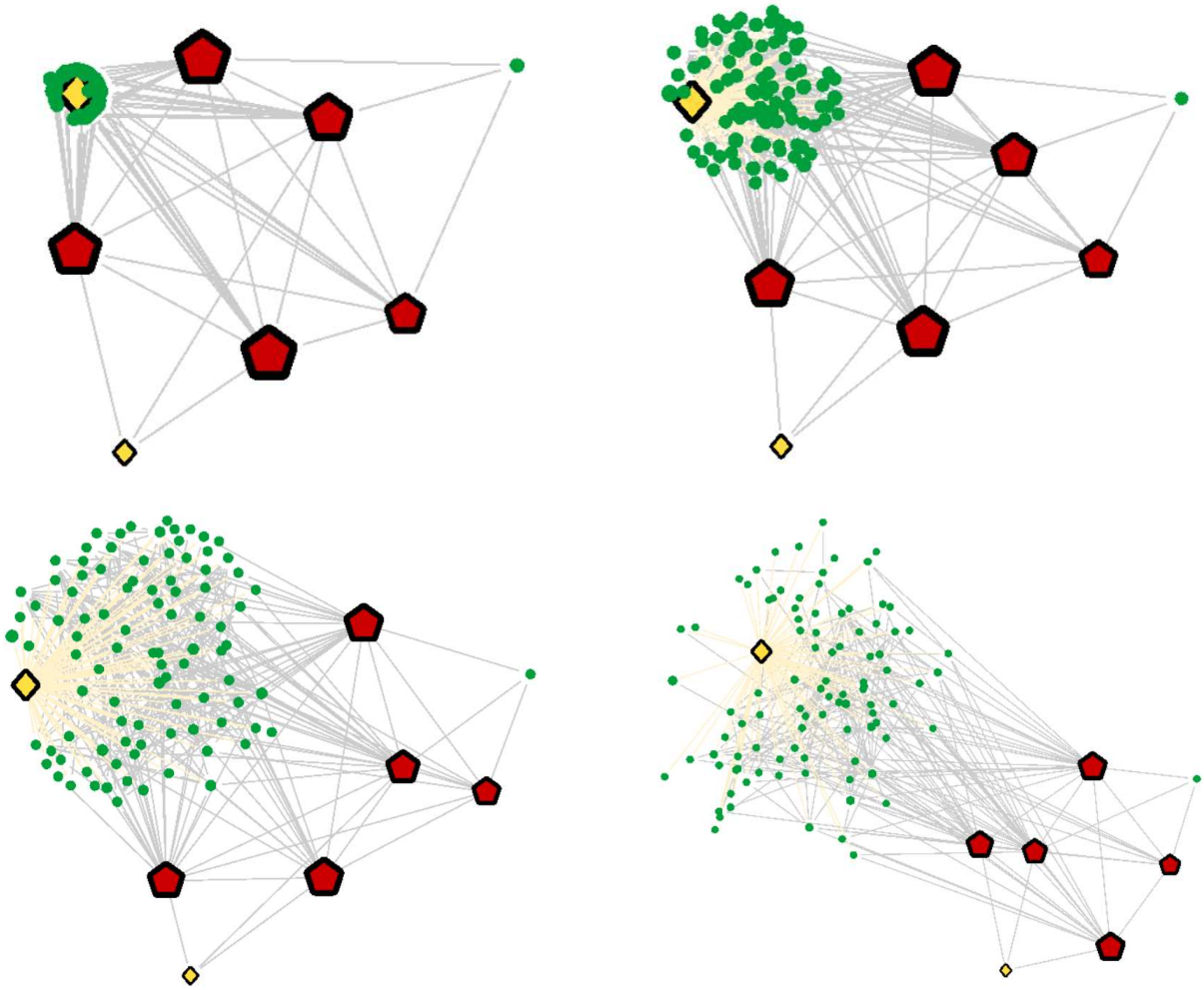


Figure 4.6: One example of the "blooming" effect occuring when unfolding a node with many children.

The comprehensibility can be improved through well chosen interaction techniques or layout algorithms. Other attempts in earlier work have attempted to minimize edge crossings or bundle multiple edges together resulting in a loss of accuracy but a gain of clarity. For the purposes of this thesis however it is decided that the decline in accuracy would negatively impact a users experience when

performing tasks like the ones described in section 6.4 so significantly, that the negative impacts would outweigh the increase in clarity. Especially when trying to find the dependencies of a node, simple straight lines are assumed to be much easier to follow. It is for these reasons that edge bundling techniques are not included in this thesis.

Figure 4.7 presents a screenshot of the *CodeExplorer*. The information bar and information box marked with the numbers one and three respectively are explained further in chapter 5. Meanwhile, the main canvas identified by the number two shows the currently visualized software graph and the number four marks the minimap.

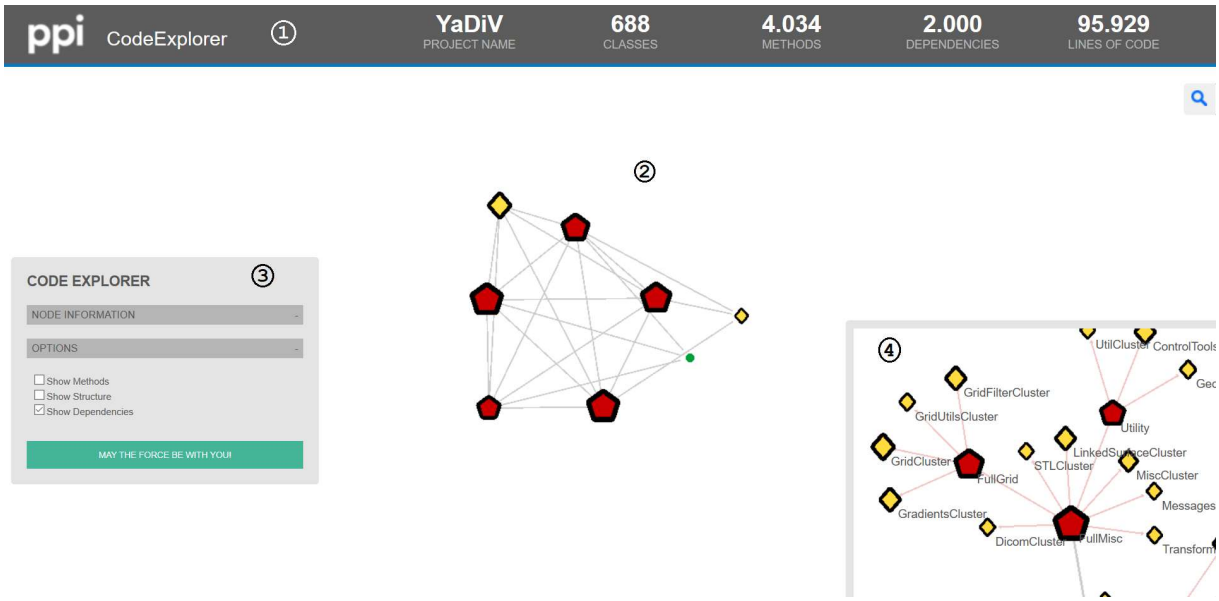


Figure 4.7: The full view of the *CodeExplorer* including changes made by this thesis.
1: information bar
2: main canvas of the graph
3: information box providing detailed information of a selected node
4: minimap mentioned in this section

Special Cases

Single Node in Two Clusters

The presented concept does not inhibit nodes being included in multiple different clusters. This can lead to some situations in which a node has dependencies to two different clusters, while upon unfolding the two clusters, only a single edge is displayed leading away from the original node.

One example present in the data used for the evaluation discussed in chapter 6 is a class called "Interpolation" with four dependencies displayed in the graph. Two of these dependencies lead to already unfolded nodes, while two others are connecting the node to the two clusters "RaycastingCluster"

and "GUICluster". Once both clusters are unfolded one of the edges disappears, while the node is now linked to the class "TransferFunctionPanel". This edge case is caused by the class "TransferFunctionPanel" being part of both clusters, while the class "Interpolation" has no dependencies linking it to other classes in either of the two parents. The described example is shown in figure 4.8.

The fact that the number of visualized dependencies decreases when unfolding nodes can be confusing to the user, especially since unfolding clusternodes generally means showing more detail. However, there is no alternative available with the current implementation to keep the user informed of which classes within an unfolded cluster an outside node is connected to. Generally speaking this edge case is heavily reliant on the visualized structure.

(a) Before unfolding

(b) After unfolding

Figure 4.8: An example for the aforementioned special case.

Folding Nodes with multiple unfolded Children

When folding an unfolded node whose children are also unfolded, the tool replicates the fold operation for all unfolded children. This means that when the same originally chosen node is unfolded again, the appearing children will be folded again instead of a large number of unfolded nodes appearing due to a single unfold operation.

4.2 Visualization of Graphs

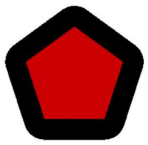
Visualizing software graphs as node link diagrams means the same visualization aspects are available for adjustment as for any other node link diagram. Due to technical details of the tool this thesis' implementation is based upon, only limited aspects are able to be changed. For the styling of nodes; the aspects of shape, colour, colour of a border and size are available. Links offer the aspects of link type, colour, colour of a border and width. Link type mostly refers to a choice between a directional or bidirectional link indicated by an appearance or absence of an arrowhead at the end of the link.

The overarching goal for the visualization in this work is to enable the user to easily distinguish and identify different types of nodes, while also pointing out similarities between nodes of similar properties. This goal is especially important since the minimap shows the same nodes that are also visible in the main graph.

As for the aspect of node colour the focus was put on the three types of nodes that are present in the graph. Aggregatenodes only contain further aggregatenodes and tagnodes, while tagnodes only ever group basic class nodes who themselves are the third and final type. Based on the idea that the ontology is the best way for an expert to add their own knowledge into the visualization, a decision is made to treat aggregatenodes as the most important type of node. These nodes include the largest amount of information as they include the largest number of nodes and therefore dependencies. Secondly, tagnodes are seen as a smaller equivalent of the aggregatenodes. They again represent knowledge gathered by experts for the visualized software. Node types can therefore be ordered in terms of their importance with respect to expert knowledge which leads to the idea of colouring the nodes in a similar way to traffic lights. More specifically aggregatenodes are assigned a red shade, tagnodes are coloured in yellow and base classes are shown in green. Red and yellow can both be seen as warning signs, indicating to the user that fold or unfold operations of these important nodes can have major impacts on the shown graph. The specific colours chosen are meant to warn the user but not become overbearing. To achieve this goal darker shades are selected.

The use of colour always bears risk in the fact that the information presented by it to the user is lost when a user suffers from colour blindness. This results in a need for a secondary way of showing the hierarchy and difference between the nodetypes. Ultimately, the node shape was decided on as the aspect used to convey the node type in conjunction with a node's colour. For the nodes representing software classes in the base graph a circle was chosen with the simple reasoning of it being a very basic shape. Additionally, the absence of corners in a circle presented an additional opportunity to distinguish between basic and clustering nodes. Aggregatenodes are assigned a pentagon shape while tagnodes are drawn as diamonds. While these two shapes are easily distinguished from one another, they are still similar enough to represent the fact that the same interaction techniques work on both types. Furthermore the amount of corners decreases in the previously picked order of importance

imposed on the three types as is visible in figure 4.9.



(a) An aggregatenode.

(b) A tagnode.

(c) A class node.

Figure 4.9: Types of Nodes visualized in the tool.

As a means of yet another distinction between clusternodes and base nodes, both clusternodes are visualized with a black border. The border helps both with the aforementioned distinction and the similarity of aggregate- and tagnodes, while also reducing the amount of colour on the screen. Less colour means that the visualization appears less vibrant and overbearing to the user, even when the tool is utilized for longer periods of time.

Link types are distinguished in dependency links, aggregatenode links and tagnode links. The former are the ones present in the base software graph and were kept as narrow directionless grey links like they were presented in the original version of the tool prior to the work done over the course of this thesis. The reason for this decision is mostly the fact that links had to be narrow in order to keep them from overlapping as much as possible while the grey colour made sure that links were not overbearing to a user. Additionally, users that had already worked with the tool before would recognize the link type based on their earlier experience. The different link types are depicted in figure 4.10.

(a) A dependency link.

(b) An aggregatenode link.

(c) A tagnode link.

Figure 4.10: Types of Links visualized in the tool.

For the other two types of links, a similar reasoning as for the two corresponding types of nodes is followed. Since the links always appear between a clusternode and its unfolded children a directional relation is present. Therefore, arrowheads were added to these types of links. Another feature styled for these cluster-child links is the colour of the link itself. To enable a quick way of recognizing which type of node any link belongs to, a yellow colour is chosen for links connecting tagnodes to their children, while red is utilized for the corresponding links connected to aggregatenodes. The specific colours are chosen based on the goal of links being visible but subtle. The user's focus should generally still be on the dependency links as they are the most important links in the graph.

Another goal of the visualization is to create visual symmetry between the nodes portrayed in the main graph and their counterparts in the ontology graph. This goal is fulfilled by using the same visualization properties for aggregate- and tagnodes in the ontology graph as in the base graph.

A different approach to the visualization of unfolded clusternodes is the idea to simply not draw any unfolded aggregate- or tagnodes. This can greatly reduce the number of visible nodes and links on the canvas resulting in a reduction of visual clutter that has less value to the user than the dependency links of the original software graph. Links between unfolded clusternodes and their children are also omitted in this variation.

One drawback of invisible unfolded clusternodes, however, is the refolding of previously unfolded clusternodes. Since they simply disappear from the graph, the node's name must be remembered and found in the ontologygraph, in which the node is still visible. The trade-off between these advantages and drawbacks is evaluated in the study described in chapter 6.

As for the traversal types described in chapter 2, the two different visualization methods offer two different types of traversals. A user is never bound to interacting with full levels of the hierarchy. Instead individual clusternodes can be folded and unfolded at any time. Therefore, the basic traversal type is the unbalanced traversal depicted in figure 2.6e. Meanwhile, the visualization method keeping unfolded clusternodes in the graph and portraying their parent-child links creates a hybrid between unbalanced and above traversal. The highest nodes in the hierarchy are always presented, while a user can still decide the fold state for each clusternode individually.

4.3 Use Cases

To give some context to the concepts developed in this thesis this section describes two potential use cases of the tool, one of which is also part of the study described in chapter 6.

Project Introduction

Alice is a new hire at a company developing software. In her first week she is introduced to a team developing a software project that she is also supposed to work on. Since the project has already been in development for multiple years there is a large amount of classes that already exist.

To help Alice gain an overview of the project and its underlying structure her coworkers advise her to use the tool *CodeExplorer* in which they have already prepared a clustering. Alice is told that all classes relevant to tasks that will be assigned to her in the first weeks at the new company are grouped in a cluster named "RelevantForAlice", in which she will find other clusternodes further separating the individual classes within this group.

Upon starting the tool Alice can first gain an overlook of the package structure of the project. She then switches over to the dependency view and is presented only two nodes on the screen with the names "RelevantForAlice" and "NotRelevantForAlice". After unfolding the node "RelevantForAlice" she can inform herself of the classes relevant to her assignments. Alice also finds that there is a central node within this part of the project that is linked to all other classes in the cluster whose source code she then looks up in the project's folder structure.

Alice thereby gets an overview of the relevant parts of the project without being overloaded with information. She can also immediately identify a central node that could be especially important and can get started more easily.

Software Migration

Bob works in the software development department of a large bank. The bank still utilizes a software backend coded in the programming language *COBOL*, which is no longer learned by many computer science students. Since the backend needs to be maintainable for years to come, the bank's management has decided that a migration of the code from *COBOL* to *JAVA* is necessary. As an expert of both the backend and its programming language, Bob is assigned to a team working on the migration.

In order to keep an overview of the project and what parts should be migrated first the backend is visualized in the *CodeExplorer*. Bob immediately identifies a few core classes in the middle of the visualization and also finds a somewhat separated group on the outside of the graph. Additionally, a few nodes also drift out of the visualization since they are not connected to any other node. Bob therefore designates these nodes as ones that need to be examined individually, but might not have to be migrated at all, since they are not sued by other parts of the software.

After some time the project team decides to start the migration by migrating the classes forming the separated group first, since changes made to these classes do not impact as many other classes in the project as changes to the classes corresponding to core nodes with high degrees would. Bob then uses the *CodeExplorer* to visualize the decided group by assigning the same tags to all classes belonging to and clustering all nodes with the tag in a clusternode. By performing these steps on multiple groups in the visualization, Bob significantly reduces the amount of nodes shown in the tool and the overall visual clutter on the screen.

At a later stage of the migration the first few classes are migrated to *JAVA*. Bob decides to group all migrated nodes as a designated cluster in the *CodeExplorer*. By folding away all migrated nodes the project team can quickly recognize which other nodes are currently linked to already migrated classes and therefore are more likely to lead to errors during tests. Such nodes connected to the clusternode also present good examples for which classes to migrate next.

5 Implementation

5.1 CodeExplorer Baseline

In order to evaluate the concepts described in chapter 4, a tool visualizing software as node link diagrams is required. The tool *CodeExplorer* developed by the *PPI AG*, a company working in the field of consulting and software development for banks and insurances, provides such a presentation. Therefore the *CodeExplorer* is used as a baseline on which the ideas discussed in this thesis are implemented.

Figure 5.1 shows the state of the *CodeExplorer* before the work done in this thesis. The most notable aspects shown are the information bar at the top, the information box on the left hand side and the main graph canvas covering the rest of the image.

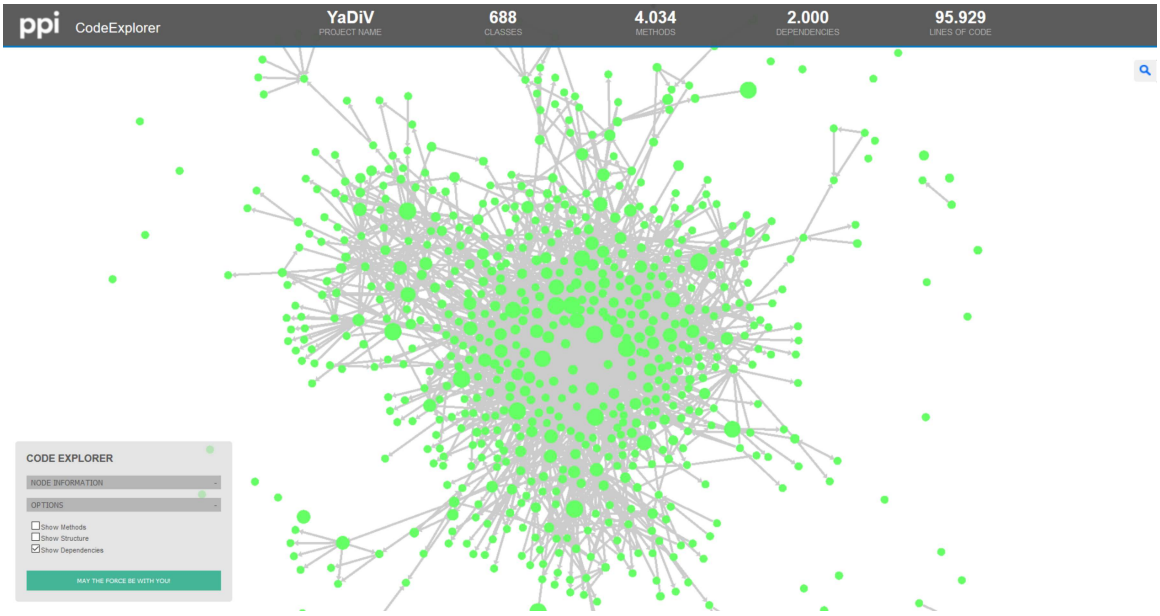


Figure 5.1: The *CodeExplorer* before the work done in this thesis.

The information bar heading the tools presentation displays some general statistics of the visualized software project. It shows the project's name in combination with the number of classes, methods and dependencies present between the classes. An overall aggregation of the lines of code metrics of all entities is also given.

Meanwhile, the information box present on the left side of the tool is designed to provide in depth information on a node selected in the main canvas of the *CodeExplorer*. When selecting a node it displays its type, name and the path under which the node can be found within the folder structure of the visualized software project. Further information is given on the number of methods, dependencies and how many lines of code are present in the file. The box also displays the *Javadoc* of the class and the tags assigned to it by an expert.

Figure 5.2: The information box when selecting a class.

In addition, the box serves as a means to interact with high level aspects of the visualization. A user can choose which aspects of the visualized software project are to be presented, most notably whether the package structure or dependency structure is to be shown. While the tool also provides options to include methods in the presentation and allows the visualization of both package and dependency structure at the same time, neither of these settings are used in this thesis. Similarly, the work of this thesis is only relevant to the graph showing the dependencies of the visualized software. Just below these options the information box finally provides a clickable button which can be used to pause or unpause the layout algorithm, meaning that movement of the displayed graph can be stopped or restarted with a simple mouse click.

The tool itself is developed in *JavaScript*, with *grunt* being used to build the project. This allows for a full folder structure to exist for the developer, while only a single file has to be recognized in the main *html*-file. The *html*-file itself can then be opened in Microsoft's *Edge*.

Using Microsoft's *Edge* to run the *CodeExplorer* brought up multiple issues caused by the browser in combination with the use of grunt regarding debugging processes which in turn had negative impact on the time needed to implement the concepts developed in chapter 4. One of these issues lies in the console output which portrayed line numbers for any errors thrown by the application. These numbers do not always reflect the correct corresponding lines in the code. There is an additional disconnect between the two versions of the file shown in the debugger of the browser and any editor used. While both are expected to display the same file, the line numbers do not always align. An even larger negative impact on the debugging of the tool is the fact that the debugger built into *Microsoft Edge* does not work as expected. Selecting a line of code in the file displayed within *Edge*, in an attempt to put in a breakpoint at that point in the code, adds the new breakpoint at a completely different part of the code. Even when able to assign a breakpoint to a desired line, the debugger executes different lines from the ones highlighted while stepping through the code. The result of these issues is that all of the debugging work on the code had to be performed using console prints, making it significantly harder and much more time consuming to find the origin of any programming errors made.

In the base version of the tool provided before the start of the implementation work of this thesis, the code requires access to two files on the machine running the tool. One is an *XML* file containing information on the software visualized in the tool while the other provides information concerning the visualization properties of nodes and edges.

The aforementioned *XML* file consists of information regarding the software visualized by the tool. It contains the package structure of the project as well as all classes present within said structure. For each class in the project the file contains some peripheral information such as any existing *Javadoc* or the lines of code metric. Most important for the purposes of this thesis, though, are the dependencies which are also present as attributes of classes. Additionally, each class can be assigned tags which are essential to the clustering conceptualized in this thesis. For the purposes of this thesis the *XML* files can be used in their provided state, however new tags were collected in an interview preceding the study and needed to be added to the individual classes in the *XML* file. Such a file can contain thousands of lines, the software visualized as part of the evaluating study described in chapter 6 for example is described in a file containing 688 classes and a total of 25220 lines of information.

Meanwhile the file designating the visualization properties is designed in a special file format named *ExpCSS*. As the name suggests it is an expanded version of *CSS* offering specific attributes for the styling of nodes and edges within the *CodeExplorer*. The file's schema is similar to the basic schema of *CSS*. It offers a way of changing the visualization attributes mentioned in section 4.2. Since the

ExpCSS file is used by a special parser to feed the information into the tool, each attribute needs to be included within the parser. As this is not the case for all attributes generally available through *CSS* files, the *ExpCSS* file structure both expands and limits the available style features in comparison to regular *CSS*. This leads to attributes like node texture being unavailable for the implementation of this thesis' concepts.

The mentioned layout algorithm was already implemented in the *CodeExplorer* as well. The tool uses the library *ngraph* as its basis for all graph related operations. *Ngraph* offers a lot of functionality in terms of interaction with graphs and is also performant with high numbers of nodes. A comparison of multiple graph libraries focussing on their computational performance can be found on YouTube[1]. The layout algorithm most prominently used by *ngraph* is the force-directed algorithm which has been found to lead to well laid out graphs while maintaining good performance even when large numbers of nodes and edges are visualized.

Other interaction methods provided by the *CodeExplorer* are a zoom in the main graph, the ability to move the canvas in all four directions available on the 2D screen and a number of interactions on the nodes themselves. A user is able to select a node by simply clicking on it. When a node is selected its dependency links are coloured differently and the connected nodes have their labels displayed. A user can therefore quickly find which nodes have dependencies linking them to the selected node. An example is shown in figure 5.3.

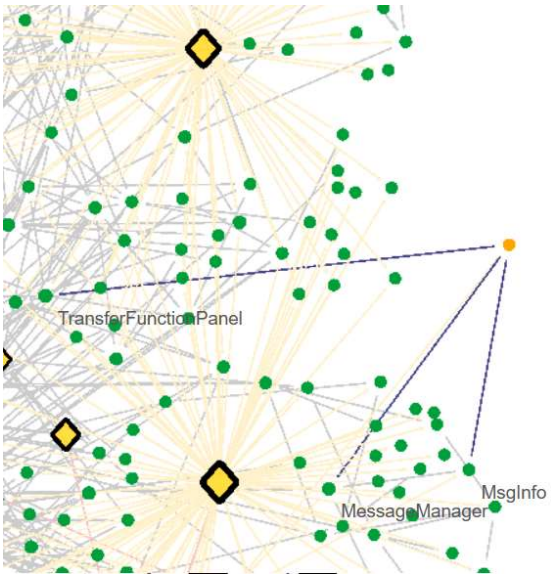


Figure 5.3: A Node is selected and its dependency links are highlighted. Additionally the labels of all connected neighbours are displayed.

[1] A video showing twelve different graph implementations including *ngraph* an be found at `https://youtu.be/Ax7KSQZO_hk`

An additional interaction method provided is a way for the user to move a node within the graph. By simply clicking and dragging, the user can change a node's location to a desired position in an intuitive way. While at this point the force directed algorithm will impact the recently moved node and therefore potentially move it back to its original position, the user also has the option to pin a selected node by pressing the key "P" on the keyboard. A pinned node will not be impacted by the forces exerted by the layout algorithm any longer, meaning that it will stay in place. Nodes that are pinned by the user can still be moved around and will remain pinned. This interaction has been found to be helpful when trying to keep a specific node in mind during the evaluation of this thesis described in chapter 6. Especially when needing to unfold multiple nodes connected to a specific entity, the pin interaction can be useful as a user requires a way to reevaluate a node's dependencies after unfolding the clusternode it was connected to.

When considering these interactions that are already present in the tool it is important to note that most of them are initiated using the mouse and not the keyboard, meaning that its keys can still be assigned functionality.

Generally speaking, the existing implementation of the *CodeExplorer* presents an extensive baseline on which this thesis' concepts can be built upon. While it limits some options in terms of visualization properties and causes a number of issues with the debugging process, it still provides interaction methods and a presentation of node-link diagrams including a layout algorithm. These aspects are detrimental to this work and would have required extensive development time which in turn would have taken away from the time spent working on the core concepts of this thesis.

5.2 Additions

The most prominent addition made to the tool over the course of this thesis is the implementation of clusternodes as described in chapter 4. Clusters defined as groups of nodes present in the base software graph first and foremost need a way of selecting specific entities that are assigned to their group. The *CodeExplorer* portrays tags allocated to entities in the information box discussed in section 5.1. They are specified for each class within the *XML* file containing information on the software project visualized in the code explorer. Within this file each class is assigned an attribute "attributes" which contains all tags assigned to it as a string separated by commas. Tags can contain any string values enabling an expert to assign any desired attribute to any given class. These properties make the tagging an obvious choice as a way to assign a class to a group. Classes that belong together in any manner could simply be assigned the same tag in order to create a grouping between them.

In an early version of the implementation discussed in this chapter tags that chosen to be grouped as clusters are simply added in an array structure within the tool's code. Any tag present in this array

will lead to all classes, that the tag is assigned to, being grouped together and a cluster appearing in the graph instead of the grouped base nodes. This means that for an array structure

```
const TEST_TAGS = ["thread", "renderer", "view"]
```

three clusternodes "threadCluster", "rendererCluster" and "viewCluster" are generated, each including all entities of the software project assigned the respective tags. Apart from this highest level of the array, clusternodes can also be grouped even further by grouping them in another array. While the former example would simply lead to three clusternodes appearing on the screen, these clusternodes can also be grouped again in order to create a hierarchy of clusters. An array structure

```
const TEST_TAGS = ["thread", ["renderer", "view"]]
```

would therefore result in an initial screen displaying only two clusternodes, namely a node "threadCluster" and another with the label "rendererviewCluster". The latter of these nodes could then again be unfolded to display the two clusternodes "rendererCluster" and "viewCluster".

This method of designating clusters presents the major drawback of having to adjust the cluster array directly within the software code. This would have meant that for every change made to the presented ontology one would have needed an expert of the visualized software to create a fitting clustering, while a *CodeExplorer*-expert would have been required to implement the changes into the visualization. Therefore the input method of ontologies was overhauled.

The new input method for ontologies revolves around another *XML* file being opened by the tool. The new *XML* file's sole purpose is to provide the information regarding which tags are to be used for clusterings and what groups are clustered together further to create the hierarchy. With a completely new *XML* file being required to fulfill this role a completely new schema can be created to fully fit the needs of the *CodeExplorer*. Ultimately, the schema is decided to make a distinction between tagnodes as nodes grouping the software entities visible in the base graph, and aggregatenodes as the nodetype governing the hierarchy creation within the ontology. After declaring a name for the ontology, the initial stage is the definition of tagnodes. For each tagnode a name, the corresponding tag to be searched for in the classes' attributes, and an additional boolean variable indicating whether or not a tagnode is to be clustered are available for adjustment. The final boolean within the tagnode definition is included to enable the user to declare special tags as part of aggregatenodes. These tags give an expert the chance to declare a tag that is passed down from the aggregatenode to its components. This means that all children of an aggregatenode are assigned the declared tag and then pass it down to their own children. For example, two clusters "cars" and "bicycles" grouped in an aggregatenode "vehicles" could both be assigned the tag "hasWheels" by simply adding "hasWheels" as a passdowntag of "vehicles". Figure 5.4 shows an example of the described situation as it would be presented in a graph visualizing the ontology. Over the course of this thesis' evaluation however, no passdowntag was ever actively used, meaning that its usefulness is still up for debate.

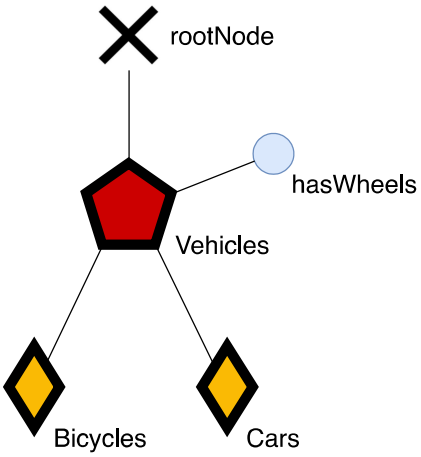


Figure 5.4: The outlined example as it would be shown in a graph visualizing the clustering. All classes contained in *cars* and *bicycles* would be assigned the tag *hasWheels*.

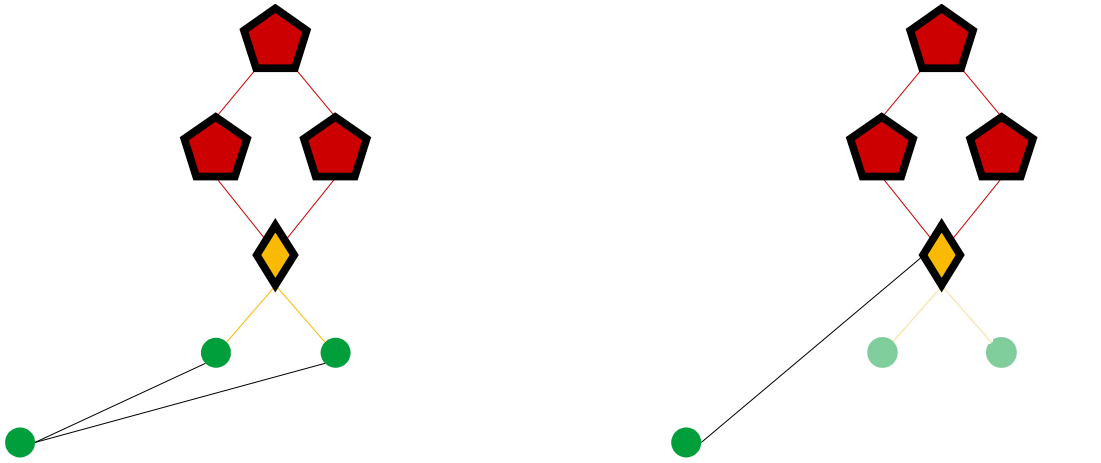
Aggregatenodes are defined with a name and a number of "aggregateComponents" who themselves declare nodes of the ontology defined in earlier lines of the XML file as parts of the current aggregation. The aforementioned passdowntags can be declared in the same way. The following example of such an XML file declares a fictitious ontology including the examples described thus far.

```
<?xml version="1.0" encoding ="utf-8"?>
<ontology name="ont_thesis_example">
        <tagNode name="threadCluster" tag="thread" cluster="true"></tagNode>
        <tagNode name="rendererCluster" tag="renderer" cluster="true"></tagNode>
        <tagNode name="viewCluster" tag="view" cluster="true"></tagNode>
        <tagNode name="cars" tag="car" cluster="true"></tagNode>
        <tagNode name="bicycles" tag="bike" cluster="true"></tagNode>
        <tagNode name="hasWeels" tag="hasWheels"></tagNode>

        <aggregateNode name="rendererviewCluster">
                <aggregateComponent name="rendererCluster"></aggregateComponent>
                <aggregateComponent name="viewCluster"></aggregateComponent>
        </aggregateNode>

        <aggregateNode name="softwareParts">
                <aggregateComponent name="threadCluster"></aggregateComponent>
                <aggregateComponent name="rendererviewCluster"></aggregateComponent>
        </aggregateNode>

        <aggregateNode name="vehicles">
                <aggregateComponent name="cars"></aggregateComponent>
                <aggregateComponent name="bicylces"></aggregateComponent>
                <passDownTag name="hasWheels"></passDownTag>
        </aggregateNode>
</ontology>
```

Based on such an *XML* file the ontology is parsed and an ontologygraph consisting of all aggregatenodes and tagnodes assembled in the hierarchy imposed by the latter half of the ontology is built. This ontologygraph is then tied together by a rootnode to which all clusternodes without existing parents are assigned as children.

To make use of the ontology in the main graph the presentation has to be adjusted massively. For each tagnode all nodes are iterated over to find nodes with the tag assigned to the tagnode. Every one of the found nodes is attached to the tagnode as a child, while also being removed from the main graph. At the same time, incoming and outgoing dependencies of each child are also saved in the tagnode itself. This information is then used to substitute the dependency links originally connecting to the clusternode's children by edges linked to the clusternode itself.

This step however can lead to issues when attempting to connect a clusternode to one of its incoming or outgoing dependencies, specifically when the node on the other side of this dependency link is already removed from the graph due to the node being part of an already clustered tagnode. In this scenario, it is necessary to find the parentnode present in the graph by moving from the originally desired node to its parent and checking whether this parentnode currently exists in the graph or not. If the parent is found, the clusternode is connected to it, otherwise the next parent is assessed in the same way. One example for such a situation is presented in figure 5.5.



(a) Original links to class nodes.

(b) Node is now linked to a folded tagnode.

Figure 5.5: An example for the way nodes are connected to folded parents of other nodes.

Another feature developed is a minimap that was added in order to show the ontology collected from the XML file. This required a second canvas to be drawn on top of the main graph and a second internal graph representation to be included. The second graph representation destined to include all relevant nodes and edges for the ontology graph caused several further problems in combination with the use of grunt, as not all files related to the renderer of the second graph were coded in a way that accounted for the existence of multiple files with the same functions being included in the same

grunt output. This cause of an issue keeping the ontologygraph from being displayed properly was especially hard to find due to the debugging issues mentioned earlier in this chapter. Once resolved however, the minimap allowed for the same pan, zoom, select and drag interactions as the main graph. Figure 5.6 shows the *CodeExplorer* including the additions made by this thesis. The minimap can be found in the bottom right corner. Additionally figure 5.7 visualizes the data pipeline of the tool including the changes made by this work.

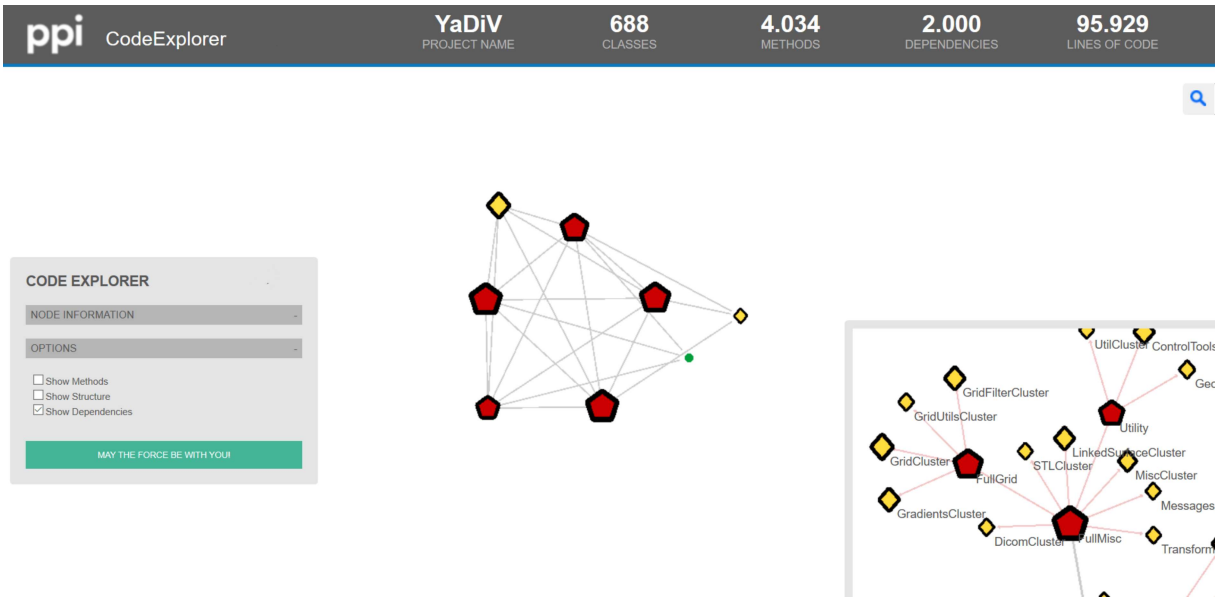


Figure 5.6: The *CodeExplorer* including the minimap in the bottom right corner.

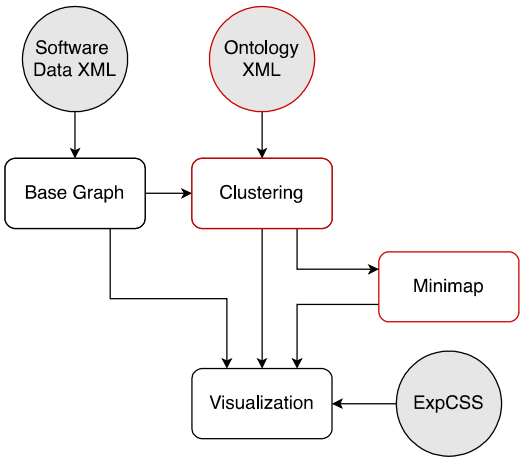


Figure 5.7: A visualization of the data pipeline of the tool. Additions made by this thesis are indicated by a red border, while circles represent files providing input data for the tool and squares represent parts of the tool's inner workings.

The most important interactions added by this work are the operations to fold and unfold clusternodes. When a clusternode is unfolded, it loses all dependency links. Its children appear in the graph and all of their original dependency links are restored. For this step cases in which the node on the other side is currently hidden in a folded clusternode are dealt with in the same way as described earlier for the initial creation of clusternodes, namely the search for a parent present in the current display. As an additional step to the unfold operation, the clusternode is linked to its children with a special link type. As mentioned in chapter 4, when a clusternode is unfolded its children are initially positioned on the same coordinates to give them a starting position beneficial to the following iterations of the force-directed algorithm.

For fold operations the steps outlined for unfold operations are simply reversed. Links connecting to the children of the folding clusternode are removed, while the clusternode itself is once again assigned the same dependency connections as when the node was created initially. Furthermore, any node representing an entity folded away within the clusternode is removed from the graph alongside the link connecting this entity with the clusternode. To enable the user to fold any unfolded node in the graph, even when children of the node are also unfolded, the fold function is simply called on all currently unfolded children, before folding the selected node itself.

A similar approach is chosen for unfold operations declared on nodes in the ontologygraph. While general fold and unfold operations work the same as mentioned above, the minimap allows for a user to unfold a node deeply nested within other currently folded nodes. Similarly to the folding of a node with unfolded children, the tool simply unfolds all folded parents before ultimately unfolding the desired node.

Choosing the key "E" for fold operations using the main graph mostly revolves around the hand placement of possible users and potential experiences with video games. The tool is mostly interacted with using the mouse, as is outlined in this chapter. Since the key "E" is positioned on the left side of the keyboard it is possible to press the key with the left hand, while a users right hand remains on the mouse. Therefore the key can be pressed without having to make significant effort of moving a hand. Additionally the "E" key is also commonly used as an interaction key in various video games which is assumed to make the interaction method and key usage more intuitive for those users who occasionally play games in their free time.

Meanwhile for fold operations on the minimap the key "R" is chosen based on the previously described choice of the key "E". Since the two interactions are very similar both in usage (selecting a node and pressing a key on the keyboard) and purpose (folding or unfolding a node), a key is chosen that is in the near physical vicinity of the key "E". Furthermore the fact that the minimap is displayed in the bottom right corner of the screen and therefore to the right of the main graph is a reason to choose the key immediately to the right of the key "E". While it would have been ideal to use the same key for both interactions, it is not technically possible.

The aforementioned keys are implemented by means of an eventlistener that waits on keyboard inputs. While it would have been ideal to issue fold and unfold interactions on selected nodes of both the main graph and the ontologygraph by the same key, the implementation of the selection of nodes means that a different key needs to be chosen for interaction on basis of the graph shown in the minimap. The issue inhibiting the use of the same key is the fact that while the tool allows for a selection of a single node for each graph, it does not record the time at which the node is selected, making it impossible to discern which node was selected most recently. Choosing a different key for interactions with the ontologygraph is the simplest solution to this ambiguity, especially since keeping one node in each graph selected at the same time can be beneficial to the user when examining dependencies of one node, while browsing the ontologygraph for one of the connected neighbours.

Another interaction implemented in the tool is the transformation of node positions according to their coordinates in the softwaregraph. The graph structure used in the "CodeExplorer" already offers functionality to set the position of a node, pressing the key "T" calls a function that makes use of this functionality. By retrieving the coordinates of nodes in the ontologygraph and iterating through all nodes present in the main graph to find the ones corresponding to entities shown in the minimap, positions can be assigned that resemble the relative positions present in the minimap. In order to keep nodes from being as close to one another as in the minimap, a stretch factor is implemented. Node coordinates are simply multiplied by this stretch factor before being assigned to the nodes in the main graph. Additionally, nodes are pinned in the main graph to keep the transformed positions.

The transformation is issued by pressing the key "T" on the keyboard. The letter is chosen based on it being a transformation technique and the fact that this choice would mean that all interactions implemented over the course of this thesis are available through presses of keys close to one another. "T" is also easily reachable with the left hand of a user, while the right hand can still remain on the mouse. Pressing the key once will initiate the transformation, while pressing it again will result in the nodes being unpinned and therefore influenced by the layout algorithm once more. This leads to them moving back into a formation similar to the one presented before the interaction, depending on whether or not fold or unfold operations were performed on the presented nodes during the time that nodes were pinned.

Design Decisions

While implementing the concepts of this thesis multiple design choices have to be made for various cases revolving around the creation of dependency links between nodes representing software classes and clusters in which such classes are grouped. One of the most obvious examples of such a decision is made for cases in which multiple children of a clusternode have dependency links to the same node. In such situations only a single edge is drawn in an attempt to reduce visual clutter.

Additionally, nodes with multiple parents remain in the graph for as long as a single parent continues to be unfolded. This decision is made with respect to the fact that a user would be confused by an unfolded cluster unexpectedly being folded halfway, because some of its nodes are part of another cluster. When a user decides to unfold a node the content of said node are of interest to them. Therefore they should remain in the graph until all parents are folded again.

Even though the ontology used for the study conducted for the evaluation of the concepts of this thesis described in chapter 6 is of a tree-like structure and each node only ever has one single parent, functionality for structures more akin to graphs is still provided. This functionality required some extensive changes, for example the search for an unfolded parent of a clusternode not present in the current graph. Since there can be multiple parents in graph-like ontologies, the closest present parent of all connected branches is linked to. An example can be found in figure 5.8.

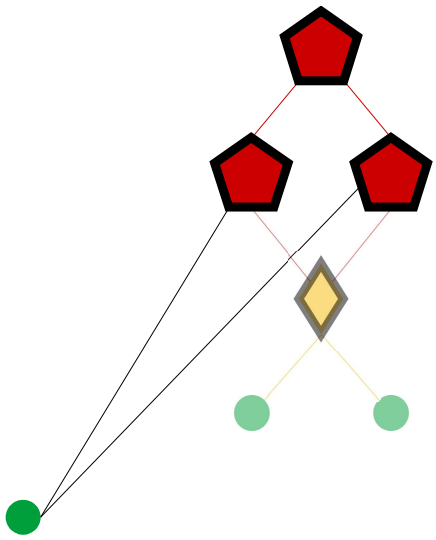


Figure 5.8: A node connecting to both folded parents of a folded tagnode. The example is the same as in figure 5.5.

For unfold operations called on deeply nested nodes in the ontologygraph with multiple folded parents the decision is made to unfold all folded parents, even if one could just unfold a single path to the desired node. The reason for this decision was the fact that it is impossible for the tool to tell which of the various paths is currently of interest to the user. All that is known is the fact that one specific node is being unfolded, no information is given on why this unfold operation is desired. Therefore all available information is shown and it is up to the user to once again fold away nodes that are not currently of interest.

6 Evaluation

In order to evaluate the value of the concepts implemented on the *CodeExplorer*, a user study is conducted. In this study participants, most of whom had no prior experience with the tool, are asked to perform tasks using the *CodeExplorer* and answer questions regarding their experience. Firstly however, a tag collection interview was held with an expert of the visualized tool.

6.1 Tag Collection Interview

Tags assigned to individual classes, and therefore individual nodes in the software graph, are the intended method for users to assign nodes to clusters. Any ontology imposed on the graph requires the according tags to be present in the visualized data. In order to find a fitting ontology for use in the study, an interview is conducted with an expert of the visualized software project.

This interview starts by asking the interviewee a few demographic questions and how well they would rate themselves to be familiar with the software project. Afterwards, an example for a potential group created by tags is given, before the interview moves on to the actual collection of tags. In addition to the specific source files of the project available to the interviewee, they are also provided the current, and therefore unclustered, visualization in the *CodeExplorer*. Note that for this interview both modes available in the tool are used, meaning that the user has access to both a visualization of the dependencies present in the project, as well as that of the folder structure.

In the case of the interview conducted in this thesis, most of the information is found by moving through the individual branches of the folder structure. This ensures that all parts of the project are considered and tagged at some point in the interview. Additionally, the folder structure gives the interviewee a way of looking at some context of the class that is currently tagged. The most detailed information about which tags belong to any specific class, however, is found in the name and source code of the file.

The tagging process took about two hours for a total of 688 classes present in the software project.

6.2 Study Objective

The overarching goal of the study, as defined by the goal definition template by Wohlin et al. [42], is to `analyze` the *CodeExplorer* for the `purpose` of improving the tool `with respect` to the cognitive load experienced by users from their `viewpoint` in the `context` of a user study conducted in this thesis.

Aspects that require major evaluation are the two visualization approaches, the different interaction methods, and the cognitive load experienced by test subjects. Additionally, the intuitiveness of the visualization is examined and a general opinion of the tool is measured.

Note that while it is not the goal of this thesis to examine the quality of either the visualized software project or the ontology imposed on it, the tool can never be evaluated without simultaneously evaluating the data used.

Overall two sets of studies are conducted, one main experiment and a preliminary study aiming to verify the questions asked. Note that for the purposes of this thesis the terms "study" and "experiment" are used interchangeably.

6.3 Preliminary Study

A validation of the study design was performed with two participants performing its first version. Insights found within this validation led to two changes being made to the study design, namely the wording of the first scenario ($\mathrm{S1}$) in the questionnaire and changes to the order of visualization approaches to unfolded clusternodes.

The first change regarding the wording of $\mathrm{S1}$ is implemented because users would take a long time to find the needed class without direct information about which clusternode it was folded in. Additionally, this wording more accurately reflects real world use cases in which a user can simply ask colleagues or would just be given the information initially.

Meanwhile, the visualization approaches are ordered in the same manner for all participants of the main study. It was found that inexperienced users would have a much easier time dealing with the visualization showing unfolded clusternodes and their parent-child links. This version of the visualization showed major benefits in helping a user understand how the clustering process itself works. Therefore the visualization omitting unfolded clusternodes is always used for the second scenario ($\mathrm{S2}$) in the updated study.

6.4 Main Study Design

Aiming to collect information on participants' opinion of both the tool in general and some of the individual visualization components, the main study of this thesis asks them to perform tasks divided into two scenarios using the *CodeExplorer*, before answering questions regarding their experience in a questionnaire. Both scenarios revolve around a new member of a software development team looking to gain an overview of relevant parts of the software project.

The experiment itself is designed with a think-aloud approach. Participants are asked to explain their thought processes loudly which in turn are recorded by the laptop included in the study. Test subjects are asked at the start of each study whether they agree to the recording or not. This decision allows for follow-up questions to be asked meaning that answers can be understood on a deeper level. Thought processes are even more valuable since a lot of questions are intentionally designed in a way to allow for some degree of interpretation. For example, one task asked test subjects to point out the key classes present in a cluster. As no definition of what makes an entity a key class is given, participants are asked about their own definition and understanding of the term. This method allows for a deeper comprehension of a test subject's thought processes and values when thinking about tasks that could be performed using the *CodeExplorer*. Note that any answers given based on follow-up questions are marked as such in the studies results in order to also preserve the original answers.

Nr.	Step	Used Materials
1	consent form regarding usage of collected data	consent form
2	collection of demographics	questionnaire
3	introduction	—
4	exploration phase	*CodeExplorer*
5	S1	*CodeExplorer*
6	S2	*CodeExplorer*
7	collection of experience with the tool and cognitive load	questionnaire
8	comments and idea regarding the tool	questionnaire

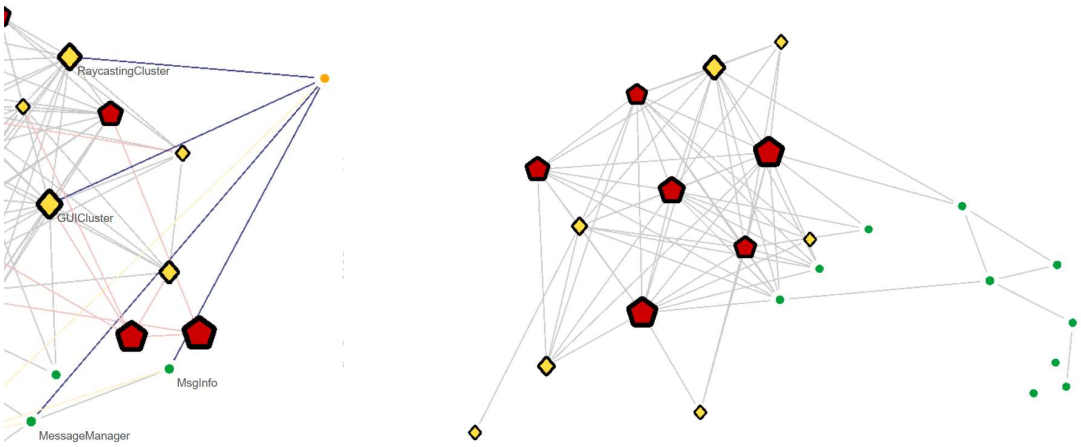
Table 6.1: Steps of the study.

Before starting to answer questions regarding the tool, participants are asked to sign a consent form regarding the collection of their data over the course of the study. This form is an adjusted version of a form used in earlier experiments operated by members of the software engineering group of the Leibniz University Hanover, similar to the one described in Klünder et al. [25].

The main questionnaire used for the thesis initially asks about demographic data of the test subject, most importantly what previous knowledge of the software project visualized in the study a user has.

Additionally, the first page of the questionnaire poses the aforementioned question as to whether a recording of the participant's laptop screen and voice is permitted.

Once a participant has finished answering this initial page, a quick introduction of the *CodeExplorer*, its purpose, and possible interaction techniques is given verbally. This introduction is intentionally kept shorter than an introduction to the tool would have been in a corporate setting, as the amount of information a participant could gather on their own without getting frustrated is also of interest. An intuitive visualization and interaction results in a quicker adoption of the tool. After the introduction participants are given five minutes to freely explore the tool and its capabilities. Any questions arising from this use would be answered depending on whether or not the response would give away answers to later parts of the questionnaire. Additionally, a cheat sheet of interaction techniques regarding the fold and unfold operations was given out.



(a) S1 (b) S2 with invisible unfolded clusters

Figure 6.1: The two scenarios worked with in the study.

After these initial parts of the experiments, test subjects are given two scenarios to work through. Each scenario consists of three questions that can be answered by use of the *CodeExplorer*. Overall, the tasks assigned to participants in the study consist of variations of the ten analytic tasks proposed by Amar et al. [4]. The tasks are designed to require the use of fold and unfold operations without mandating which exact interaction needs to be used. In basic terms, participants need to find a node within the main graph based on a name of the represented class and a clusternode in which the node is folded. Afterwards, participants are required to examine the dependency links of the found node and unfold clusternodes to which it is linked. Lastly, the contents of unfolded clusternodes need to be explored. Figure 6.1 shows two scenarios that participants are asked to work through, with S1 describing a use case in which a user is asked to make changes to a single class whose node can be found in the clustered software graph as shown in figure 6.1a. S2 concerns all nodes that are part of a specific cluster depicted in figure 6.1b.

Note that while simply giving a test subject the names of the clusternodes containing the relevant entity removes some exploration, only the names of leaf nodes within the ontologygraph and therefore the names of clusters directly containing the node are disclosed. Additionally, an exploration of the ontology would have put more emphasis on an examination of the quality of the data, rather than an examination of the tool itself. Ultimately, it is also imaginable that the same information would be given in an actual use case.

Another important detail of the study is that in between the two scenarios described to the participant the visualization is changed. For the duration of $\mathrm{S1}$ the tool shows unfolded clusters in the main graph who are connected to their children through parent-child links as described in section 4.2. $\mathrm{S2}$, however, is worked on with unfolded clusters and parent-child links being omitted from the visualization. While they are still part of the graph structure and thereby add forces to the force-directed algorithm, they are not visible to the user any longer.

The final part of the experiment consists of a questionnaire asking the participant questions regarding the usability of the tool. Answers are asserted via Likert scales indicating a level of agreement to a statement issued within the questionnaire. The scales mostly consisted of five different possible answers ranging from full disagreement to full agreement. For statements regarding the transformation of node positions issued by pressing the key "T" on the keyboard, an additional answer indicating that the interaction had not been used at all is added. Questions are asked concerning the use of the two visualizations of unfolded clusternodes and the minimap. Participants are also queried on the meaning of both the colour scheme and the size of clusternodes and their respective meanings in the tool.

Evaluations of tools like the one created in this thesis have introduced another concept to designers, namely the cognitive load. This concept is a measure of test subjects' effort when performing tasks with the evaluated tool. It is supposed to complement the evaluation previously conducted through measures like task completion speed. In other evaluations of visualization systems, for example in Huang et al. [22], this load has been tracked by self reporting and eye tracking measures. Since eye tracking measures can not be implemented due to the COVID-19 pandemic, the cognitive load experienced by the test subjects is examined by means of the Paas scale proposed in Paas et al. [34].

Finally, room for feedback on what aspects are still missing in the visualization according to the participant is given and they are asked whether or not the tool is useful for an introduction to a software project.

The questionnaire can be found in the appendix of this work. Due to all test subjects speaking German as their first language, both the questionnaire and the recordings are also in German.

6.5 Participants

Studies are conducted with a total of ten participants aged between 21 and 48 with a mean of 26. All test subjects identify as male and while three of them are business professionals in the field of computer science, the other seven are university students of related lines of study.

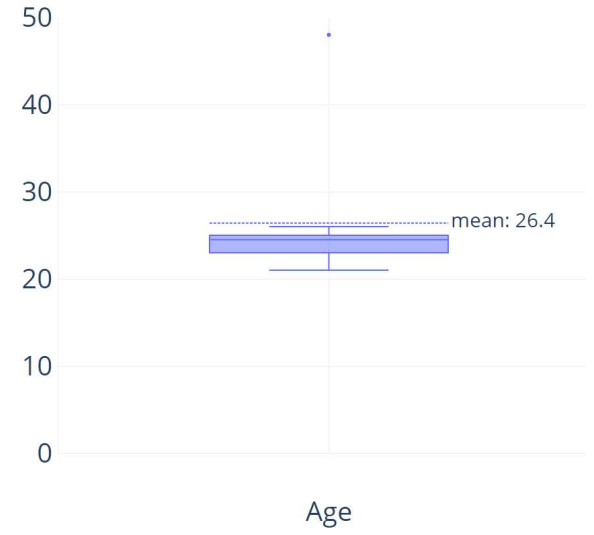


Figure 6.2: A boxplot presenting the age of participants.

All participants took between 41 and 63 minutes with a mean of 53:42 minutes (SD = 7:46 minutes) to complete all tasks and the questionnaire. More details can be found in the corresponding boxplot in figure 6.3. A recording of both the screen and audio was accepted in all cases.

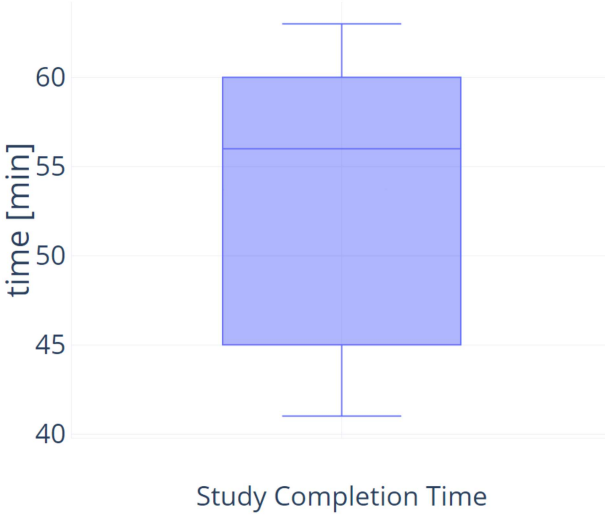


Figure 6.3: A boxplot presenting the time required to complete the study.

Four participants had prior knowledge of the software project visualized within the tool, one of which even designated himself as the software's main developer.

Prior knowledge regarding the use of the *CodeExplorer* was indicated by two participants who themselves have done programming work for the tool. However, their work is unrelated to the concepts outlined in chapter 4. Other test subjects had seen the tool in earlier stages of development but had never interacted with it themselves.

It should be noted that as is visible in figure 6.2 most participants are still young. This means that they are especially likely to find themselves in situations similar to the ones outlined in this study, as they will likely start working as developers in new software projects and teams in the near future.

6.6 Restrictions due to COVID-19

Due to the fact that the tool is worked on using a company laptop and the tool itself being owned by *PPI*, participants are required to come to an office building to take part in the experiment. This leads to complications caused by the COVID-19 pandemic, as social distancing is mandated by company standards. The most prominent risk presents itself in the peripherals of the laptop, specifically its mouse and keyboard, as every single participant has to come into direct contact with them in order to perform tasks within the study. Complying with company standards each participant who is not a direct employee of *PPI* is asked to fill out a form regarding information on whether the person had previously been in contact with others affected by the disease, as well as other pandemic-related questions. As an additional precaution, all peripherals, pens and office furniture are wiped down with disinfectant before and after each iteration of the study.

An experiment design proposed by Perer et al. [35] includes 20-60 participants and sessions of 1-3 hours in length. While such a design would have improved the study overall, it was not practical due to the pandemic. Similarly, methods of measuring the cognitive load experienced by participants using eye tracking were not possible.

While it would have been safest to conduct the study online, it is simply not practical for this particular study. For security reasons a participant can not be allowed to seize control over the company owned laptop through use of a tool like *TeamViewer*. Neither is it possible for a participant to direct a mouse operated by researchers as mouse movement needs to be precise when using the tool and inherent researcher biases can not be avoided in such a setting. A last resort would have been to conduct the study using screenshots and asking a participants to explain what is being presented to them but the results of such a study design would have been significantly less indicative of the strengths and weaknesses of the evaluated concepts.

At the time the experiment is conducted the pandemic's severity has declined considerably which results in the proposed study design to be deemed safe enough to conduct the study in its original form, as long as the aforementioned precautions are implemented.

6.7 Used Materials

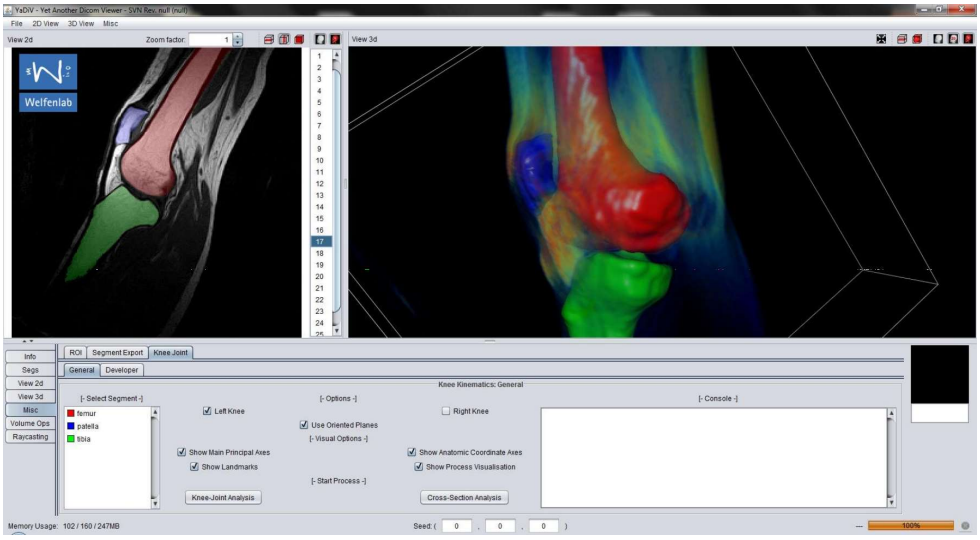


Figure 6.4: A screen within *YaDiV* showing the visualization of a knee.

The software project visualized in the tool for the purposes of the study is called *YaDiV*, an acronym for "Yet Another DIcom Viewer". It is a tool for an interactive visualization of 3D volumedata in the medical field. *YaDiV* reads data in the *DICOM* file format, specifically *DICOM* file sets, and contains models for various purposes like

- 2D visualization
- 3D volume visualization (based on both 2D and 3D textures)
- 3D segmentation
- 3D segment visualization (based on both 2D and 3D textures)
- support of stereographical visualization and haptic input devices

The tool was developed from 2005 to 2017 by Dr. Karl-Ingo Friese and students of the "Welfenlab" of the Leibniz University Hanover. It is currently being used as a platform for research in the field of 3D data processing in a medical context.

7 Results

7.1 Results of the Study

7.1.1 Preliminary Study

As mentioned in chapter 6 a short preliminary study was conducted in order to verify the experiment design. The results of this study led to changes to the wording of the first task and the choice to not change the order of the two visualizations. This means that while some tasks were impacted by changes, most questions answered in the questionnaire were not. Therefore, most answers given in the preliminary study are also included in the general study results. Another indication for the validity of answers in the preliminary study are the comments made by participants present in the recording. For example, reasons mentioned for the rating of the final question of the survey asking about whether or not a participant would recommend the tool for introductions to a new software were very similar between the preliminary and the main study.

An example for gathered information that can not be compared to data originating from the main study is the task completion time of the different tasks, especially of $\mathrm{S1}$. Since test subjects in the preliminary study were not given the name of the cluster containing the node of interest, they spent much more time searching for the node than participants of the main study.

Two test subjects took part in the preliminary study. Their task completion times can be found in figure 7.1. The two ratings given for the cognitive load were the highest measured across all study participants with values of eight and seven. Similarly, their rating of the usefulness of the visualization hiding unfolded clusternodes was especially negative with values of two and one.

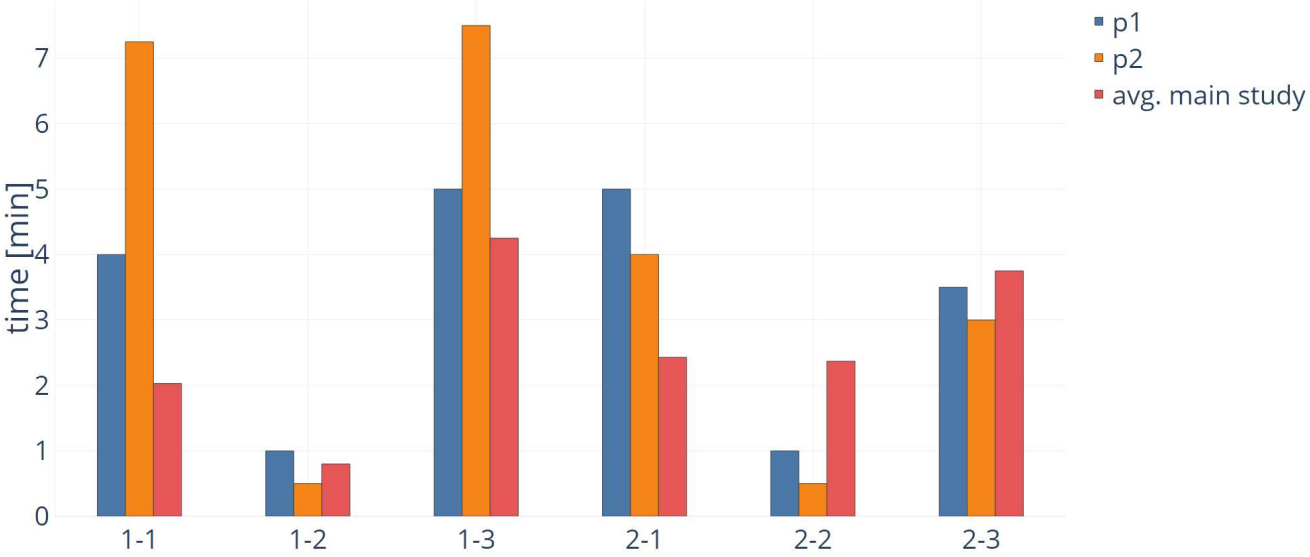


Figure 7.1: Task completion times in the preliminary study by task and participant. The average task completion time for all tasks measured in the main study is also depicted.

7.1.2 Main Study

Scenarios

Task completion times were measured for every question answered with use of the *CodeExplorer*. The two scenarios presented to participants each consist of three tasks, meaning that a total of six measurements were collected for each participant. Note that while test subjects were told that task completion times were part of the gathered data, they were also informed that the goal of the study is not to complete all tasks as fast as possible. Instead the focus was put on the thought processes and the results found.

The very first task of $\mathrm{S1}$ concerns the clusters a user has to unfold in order to make a specific node of interest appear in the visualization. Participants need to locate a specific cluster and unfold it, then write down all unfolded clusters. Across the eight participants of the main study task completion times ranged from one minute to 3:30 minutes, with a median and mean of two minutes (SD $=$ 42 seconds). A boxplot of all times measured can be found in figure 7.2.

As a second task, test subjects were asked to name all clusters the newly located node is connected to, before being instructed to also point out which entities of the base software graph the node is linked with. Corresponding boxplots of measured timings can be found in figures 7.3 for tasks two and three.

With $\mathrm{S1}$ completed, $\mathrm{S2}$ was introduced along with a different visualization. In the new scenario participants were asked about relevant classes for a task. While this question is open for interpretation

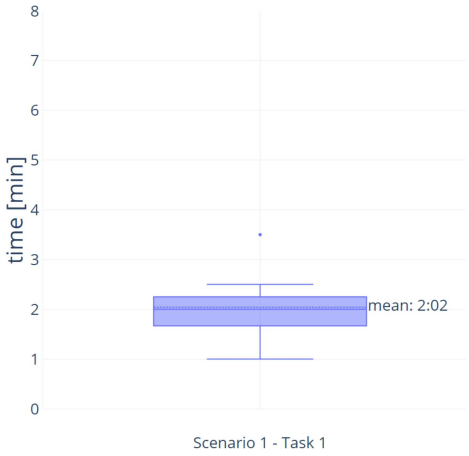


Figure 7.2: Task completion time for the first task of S1.

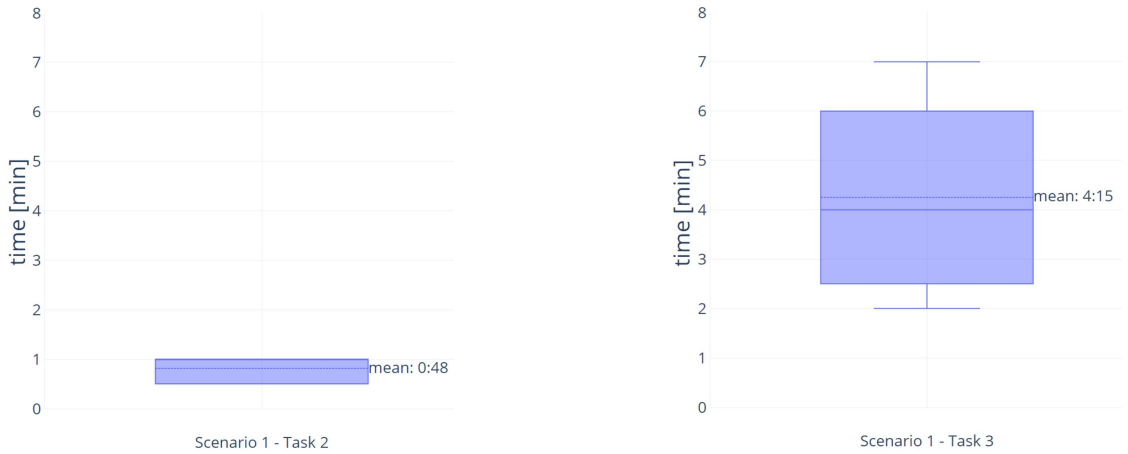


Figure 7.3: Task completion time for the remaining tasks of S1.

and leads to answers differing between test subjects, the focus is on the user's thought process leading to their result. Therefore answers that do not align with the solution prepared by the researchers are still correct as long as the test subject's thought process is reasonable for the given assignment and their interpretation of the question can sufficiently be answered by use of the tool.

Task completion times of the first exercise of S2 ranged from 1:30 minutes to 3 minutes with a median and mean of 2:30 and 2:26 minutes respectively. A boxplot is presented in figure 7.4. Measurements for the two remaining questions in the scenario can be found in figure 7.5.

While question two of S2 concerned the search for "key" nodes within a given cluster, question three combined the latter two questions of S1 into a single task. Completion times can therefore be compared especially well between the first tasks of both scenarios and between the combination of the two latter tasks of S1 and assignment three of S2.

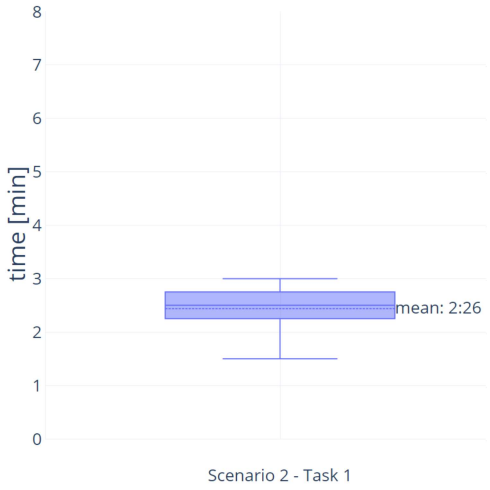


Figure 7.4: Task completion time for the first task of $\mathrm{S2}$.

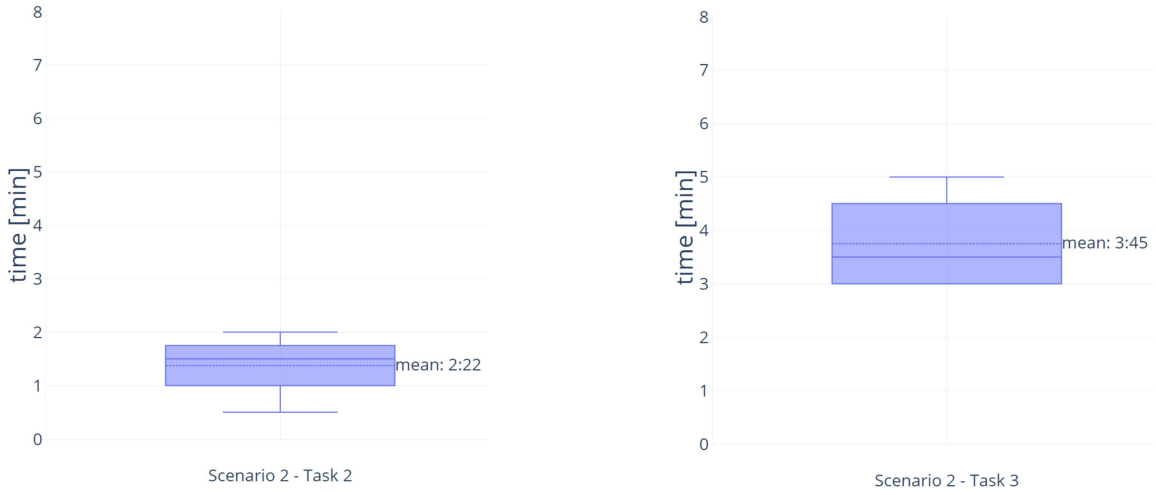


Figure 7.5: Task completion time for the remaining tasks of $\mathrm{S2}$.

The first task of both scenarios took a little longer in $\mathrm{S2}$ which is at least partially caused by the answer requiring a user to write down more names. Meanwhile, the measured task completion times of the assignments regarding the dependencies connecting a specific node to both other clusters and other classes are marginally lower in $\mathrm{S2}$ in comparison to the first. Both differences between sets of task completion times are not statistically significant ($p = 0.23$ and $p = 0.16$ respectively). Note that these values are calculated with the data of the eight participants partaking in the main study.

While the tasks are comparable in terms of the steps that a user has to take in order to complete them, it is worth noting that the variance in measured timings is also impacted by both the changed visualization and the learning effect caused by the time spent using the tool. The learning effect is especially important since test subjects not only gather knowledge of the *CodeExplorer* and its usage but also of the data visualized within the tool. Both information on the tool *YaDiV* presented as a

software graph and on the ontology imposed on the graph can be useful for tasks assigned in later parts of the study.

The learning effect does not only impact the time required to complete a task but also the quality of answers. As is shown in figure 7.6 answers given for assignments of $S2$ were correct more often. However, even for the first task most users were able to find the answers they were looking for.

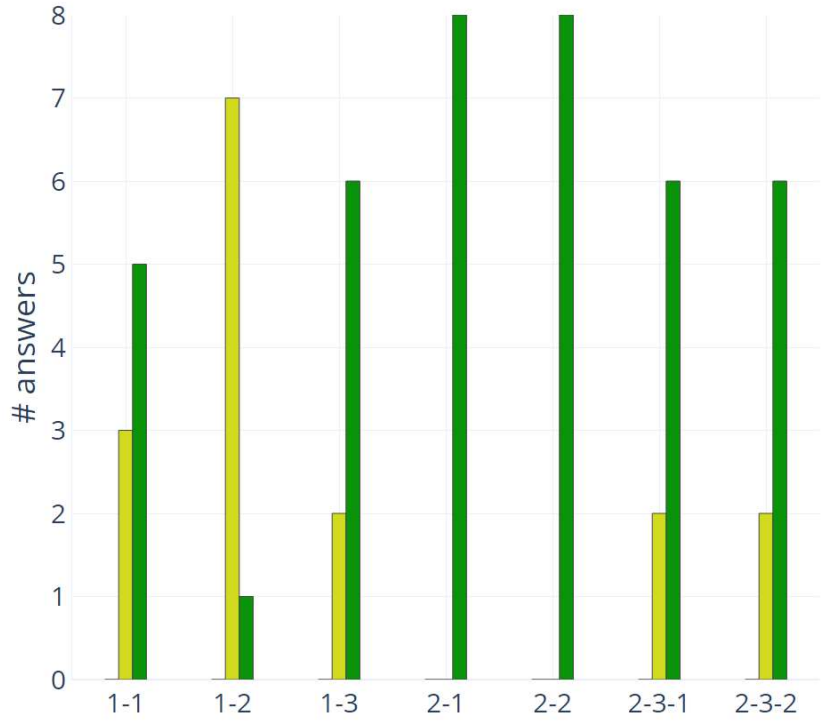
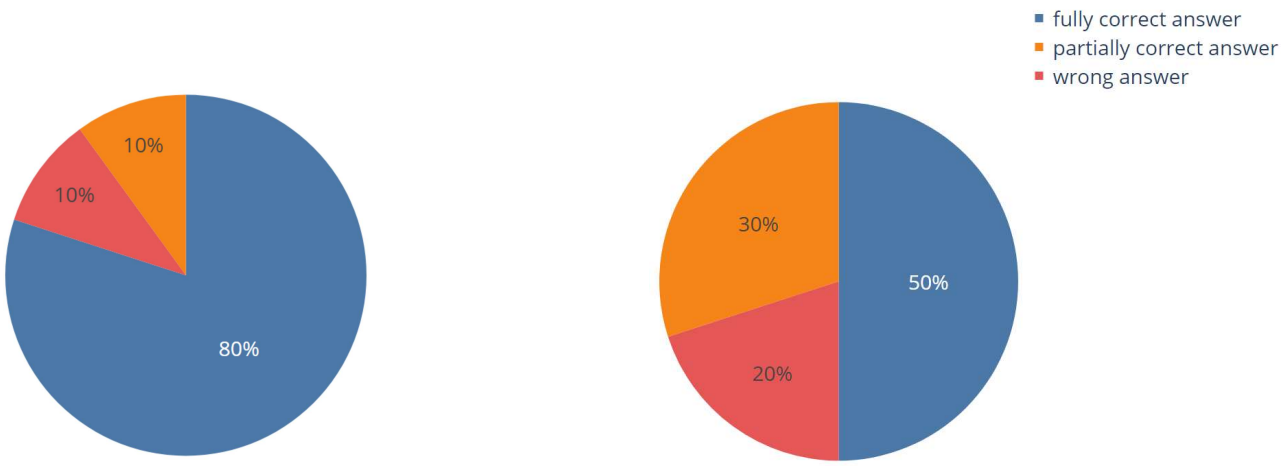


Figure 7.6: Task correctness coded as wrong answers in red, partially correct answers in yellow and fully correct answers in green. Only results of the main study (n = 8) are visualized. Note that for task 2-3 answers indicating clusters and classes are separated into two columns labelled 2-3-1 and 2-3-2.

Experience with the tool

After completing the scenarios all ten participants of both studies were questioned on their experience with the tool. In addition to multiple Likert scales measuring opinions on various aspects of the visualization, they were also asked about the perceived meaning behind visual attributes of nodes, namely their size and colour. While many participants had raised questions on the meaning of colour of nodes, 80% of them interpreted them correctly after using the tool as can be found in figure 7.7a. Figure 7.7b presents the correctness of responses to the meaning of the size attribute of nodes.



(a) Ratings of answers concerning node colour.

(b) Ratings of answers concerning node size.

Figure 7.7: Correctness of answers regarding attributes of visualized clusternodes.

Since answers to questions with interpretable wording are often hard to compare, a coding system is implemented for answers to all tasks of both scenarios and the questions regarding visual attributes of nodes. Scores ranging from zero to two are assigned to each answer with a score of zero indicating a wrong answer. Scores of one and two indicate that the answer is considered at least partially correct, with the highest number being used for completely correct answers. Note that for answers to tasks performed using the *CodeExplorer* are also coded as completely correct if the user found the information they desired according to their own interpretation of the question as long as this interpretation was reasonable.

An example for such an answer that was still coded as correct is a participant who chose the key nodes of the cluster in task two of $S2$ based on both the amount of dependencies connected to nodes and the names of the represented classes. This resulted in additional entities being indicated as key components. While the answer therefore did not fully align with the results envisioned by the researchers, an indication of the thought processes leading up to the choice of the entities allowed for a concrete decision of whether a test subject had found all entities relevant to their definition of a "key class" or not.

Most of the remaining questions in the questionnaire use Likert scales indicating agreement or disagreement with a statement. These statements concern various aspects of the visualization and are used to gather a user's opinion on their respective usefulness.

The first statement regards the two versions of the visualization used over the course of the study. Participants were asked to indicate whether or not they preferred the visualization with hidden unfolded clusternodes. As visualized in figure 7.8, a clear preference could not be found. Instead multiple test subjects expressed that while hidden clusternodes greatly reduced the amount of visual

clutter on the screen, the visualization also made it harder to keep track of changes made, especially when trying to undo unintended unfold operations. One participant even went as far as saying that both visualizations have their place in the tool with the one showing unfolded clusternodes being suited for pure clarity and beginners using the tool, while the second one is more useful for experienced operators. Note that for all results of Likert scales a rating of five indicates full agreement with a statement, while a rating of one indicates full disagreement.

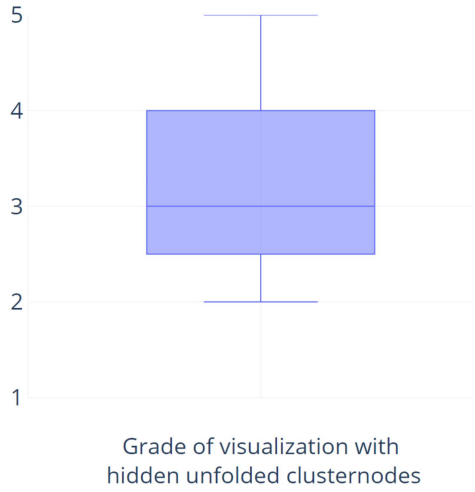


Figure 7.8: Ratings indicating agreement or disagreement with a statement that the visualization with hidden unfolded clusternodes is better than the one presenting them.

A clearer result is present regarding the helpfulness of the minimap. Figure 7.9 shows that most test subjects fully agreed with the statement that the minimap is helpful for the tasks performed in the study. With a median of 4.5 it is quite clear that the minimap was seen as a positive addition to the tool.

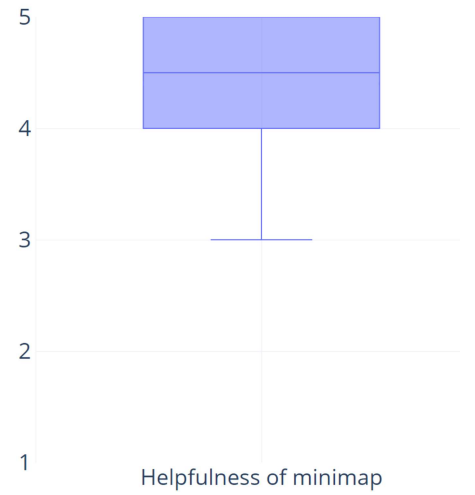
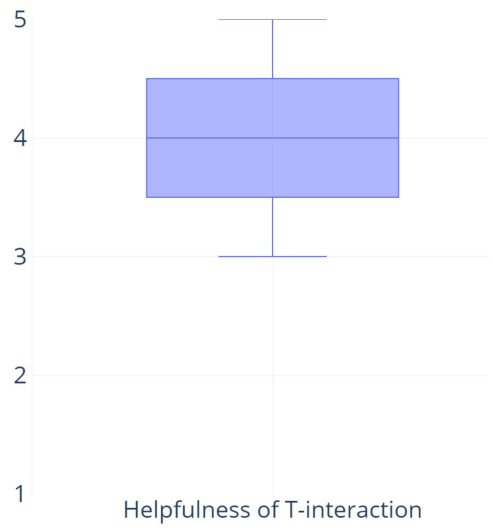
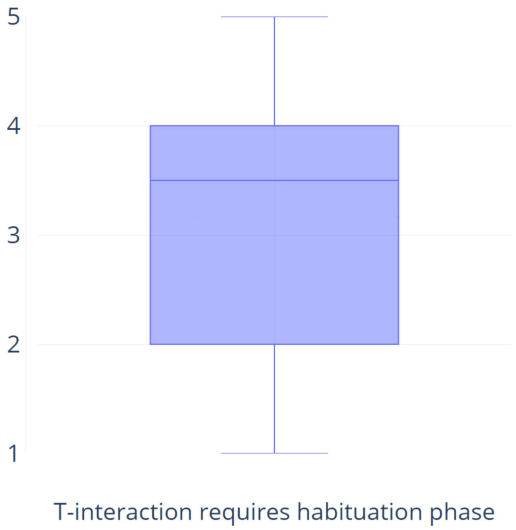


Figure 7.9: Ratings indicating agreement or disagreement with the minimap being helpful.

As for statements regarding the interaction transforming the positions of nodes in the main graph according to their position in the ontology graph, only four participants indicated that they had used the tool at all. Their answers are depicted in figure 7.10. A median of 4.0 for answers asking about the helpfulness of the interaction indicates the notion that the interaction can be helpful, even though the sample size is quite small. The second statement regarding the interaction asked about whether or not participants believed that a habituation phase was necessary for the interaction. In addition to the four participants who indicated that they had used the interaction during the study two further participants also answered this question leading to a median of 3.5. It is also noteworthy that most participants who used the interaction in their work on the scenarios indicated lower need for a habituation phase than the ones who answered the question without having utilized the interaction during earlier parts of the study.



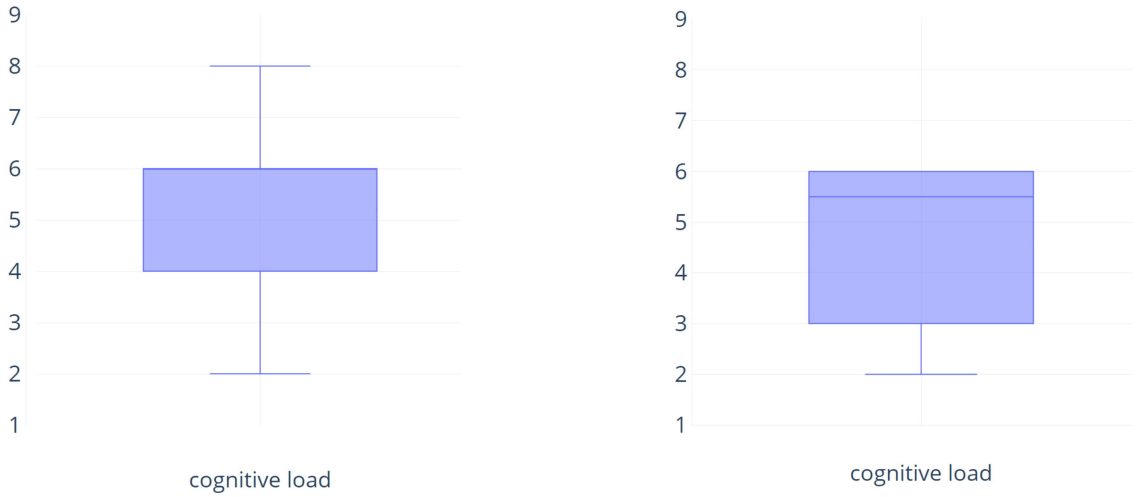
(a) Helpfulness of the interaction.



(b) Need for a habituation phase.

Figure 7.10: Agreement or disagreement regarding statements on the interaction transforming node positions.

Similarly to the two statements made regarding the interaction transforming the position of nodes, the cognitive load mentioned in earlier parts of this thesis was also measured using Likert scales. Unlike other questions asked in the questionnaire however, the measurement of cognitive load following the work by Paas et al. [34] used a nine point Likert scale and did not rely on a statement that was agreed or disagreed with. With a median of 6.0 the cognitive load shown in figure 7.11a varied heavily between participants and overall did not indicate a particularly high or low cognitive load. It is noteworthy however, that these values include the answers of the preliminary study that were the highest recorded answers. When only recognizing measurements of the main study the median changes to a rating of 5.5. The boxplot for this subset of values can be found in figure 7.11b.



(a) All values.

(b) Only responses from the main study.

Figure 7.11: Results of the Paas Scale according to Paas et al. [34].

The other Likert scale regarding the cognitive load asked for an opinion of whether or not the cognitive load would decrease after a longer habituation phase with the tool. A minimum rating of 4 shows that every single participant including the ones taking part in the preliminary study, agreed that the cognitive load would decrease over time. A corresponding boxplot is presented in figure 7.12.

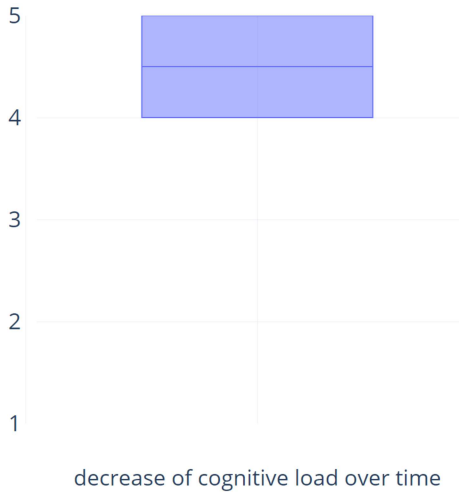


Figure 7.12: Ratings indicating agreement or disagreement with a statement regarding the experienced cognitive load.

A final Likert scale asked about agreement or disagreement with the statement that the tool can aid users with the introduction to a new software project. Answers to this question were the most positive out of all questions asked with a median of 5.0. Figure 7.13 shows the boxplot containing the answers of all participants.

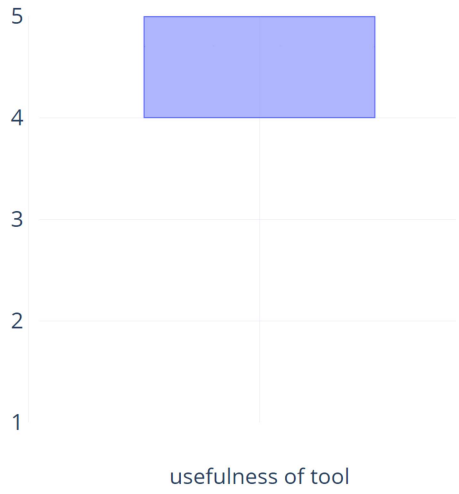


Figure 7.13: Ratings indicating agreement or disagreement with a statement regarding the usefulness of the tool.

As for data recorded in the background of the study, further findings are the usage statistics of the fold and unfold interactions. While the usages for nine participants are presented in figure 7.14, the data for one last participant was lost. Within the recorded data not a single participant used the double click interaction within the minimap. Instead every single participant made use of the key "R" in order to unfold clusters selected in the minimap. As for interactions with clusternodes presented in the main graph, only a single participant used both double clicks and the key "E" to fold and unfold nodes. Three other participants only used double clicks while 5 others exclusively used the key "E" to issue such operations.

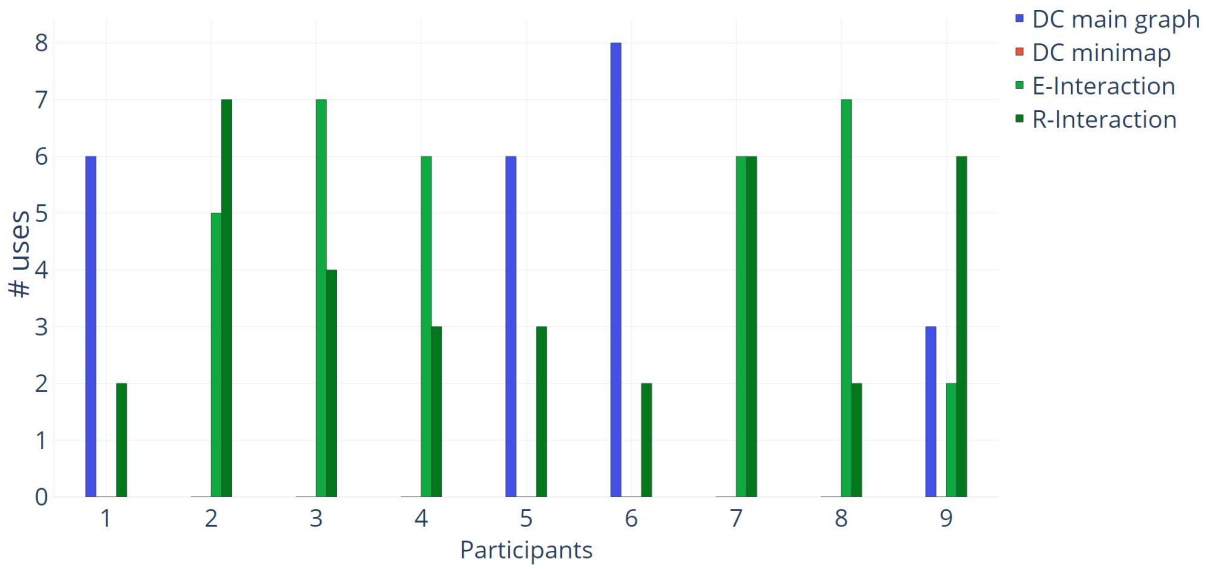


Figure 7.14: Uses of the different interactions visualized for each participant.

Ideas and Comments

With the study being designed as a think aloud experiment participants could voice their ideas and comments on the visualization and the tool itself at any time. Some of the expressed statements regarded the general visualization of the tool, others concerned the additions made by this thesis specifically. While not all comments are relevant for the evaluated concepts, they can still be helpful to the general work on the *CodeExplorer*.

One idea independently mentioned by multiple participants was to enable an interaction with a single node allowing a user to make it a focus point of the tool. While some users worked around the absence of this functionality by pulling a node out of the densely populated area of the graph and pinning it as shown in figure 7.15, it would be simpler and more intuitive to just indicate a focus node by changing its colour, size and/or texture.

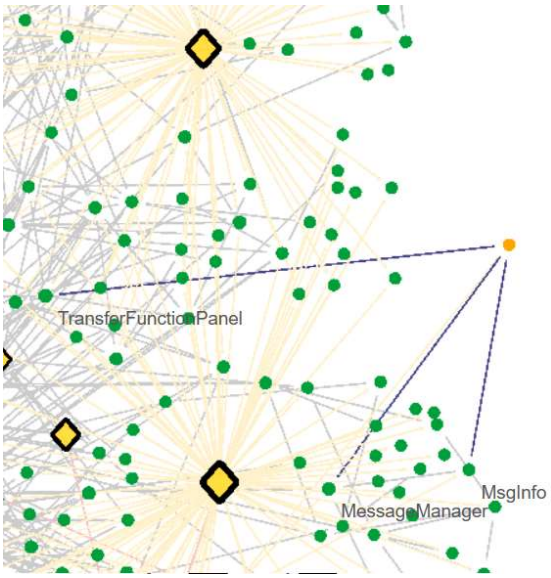


Figure 7.15: A node is pulled away from the main graph and pinned.

Similarly, a visual indication of the fold state of nodes in both the main and the ontology graph was brought up as desirable. While the visualization variation that removes unfolded clusternodes and their parent-child links partially fulfils this desire, a visual difference between nodes that are currently present in the main graph and nodes that are still folded away in other clusters or unfolded clusters would still improve the usability of the minimap significantly.

Another interesting finding of the study that was indicated in the comments made by participants is the fact that it was not always clear that differently coloured links have different meanings. Namely the parent-child links added by this thesis were not always interpreted as such and instead were mistaken for dependencies.

It is also noteworthy that multiple participants expressed at some point that they originally had assumed clusters to be distinct from one another, which led to minor levels of confusion when a node was found to be part of two different clusters in the visualization used for the study.

A comprehensive list of comments and ideas expressed over the course of the study can be found in the appendix.

7.2 Meaning of the Results

The results of the preliminary study heavily indicated that the visualization hiding unfolded clusternodes is not suited as the initial variation presented to a beginner using the tool for the very first time. This led to the study always using the visualization presenting unfolded clusternodes and their parent-child links for $\mathrm{S1}$.

Concerning the lack of a statistically significant difference in task completion times between the two scenarios, it is noteworthy that the measured times are impacted by a multitude of factors. Stemming from the tasks themselves differences occur due to the tasks not being the exact same. Additionally, some users also delayed their task completion by voicing comments. Therefore, a large amount of variance is expected especially when taking the low sample size into account.

Another finding in the study is the fact that without a direct explanation of the types of clusters present in the visualization, it was not always clear to users that when a node has a link to an aggregatenode in the graph, it is also connected to other clusters apart from the ones currently shown in the presentation. This led to many participants not unfolding aggregatenodes when asked which clusters a specific node is connected to. While responses to this type of question more often mentioned all connected clusters in the latter half of the study, 25% of test subjects of the main study still only mentioned the clusters initially shown in the visualization during $\mathrm{S2}$.

Generally speaking, the results of the Likert scales and comments regarding the two visualization types indicated that the variation with hidden unfolded clusternodes is better suited for experienced users but greatly reduces the amount of visual clutter on the screen.

As for the minimap, all but one test subject indicated that they found its addition to be helpful to the completion of their tasks, meaning that the ontology graph can definitely be seen as an addition worth keeping in the tool.

The interaction transforming the position of nodes in the main graph according to their positions in the ontology graph was seldomly used but appeared to be helpful to users who did utilize the functionality. While no clear indication was found regarding participants' opinion on the need of a habituation phase with the interaction, there are further changes to be added to the interaction that could greatly improve its use. These potential additions are discussed in section 9.2.

Visual attributes of nodes were mostly interpreted at least partially correctly. While 80% of test subjects figured out the exact difference indicated by nodes' colouring, 30% of answers regarding the meaning behind the size of nodes were partially correct with another 50% giving the fully correct solution. The 30% not arriving at the fully correct conclusion were simply including the amount of clusternodes in their metric deciding the size of clusternodes, while in fact only the base nodes were counted. These results suggest that while the exact metric was not always fully clear to the user, the general indication given by the visual attributes of nodes was mostly interpreted correctly.

The average cognitive load measured is neither particularly high nor particularly low. This indicates that the tool does require the user to pay attention, but does not overwhelm them. The fact that the cognitive load measured in the preliminary study was also significantly higher than the average load measured overall indicates that the changes made to the study design were successful in making the study as a whole more approachable, while not impacting the authenticity of the presented scenarios. Only the removal of the impact of learning effects on the results of the two different visualization variations was accepted as a negative influence caused by the adjustments.

When comparing the usage of double click interactions to the usage of the "E" and "R" keys, the results show that users mostly focus on a single technique when interacting with the main graph, while double clicks are never used in the minimap. This means that both interaction methods for the main graph are useful for the tool while double clicks on the minimap seem obsolete based on the results of this study.

Finally, the rating of the tool's overall usefulness for introductory processes in new software environments shows that most users find the tool helpful. Multiple test subjects commented that the main reason for their response was the fact that the tool offers a quicker and easier overview of the software code. The comparison to the process of opening individual source code files and searching for its dependencies to repeat the process for the dependent classes was mentioned as especially favourable. The results of the study therefore suggest that there is a distinct need for a tool like the *CodeExplorer*.

8 Discussion

8.1 Interpretation of the Results

Participants of the study mentioned on multiple occasions that the visualization variant omitting unfolded clusternodes from the graph meaningfully decreased clutter on the screen, but was also significantly harder to use. The variation is therefore seen as useful for experts, but harmful to the experience of beginners using the tool for the first time. It is important to recognize that an "expert" in this context is not necessarily an expert of the *CodeExplorer*, but rather an expert of the software visualized within the tool. A user that is familiar with both the software visualized and the ontology imposed onto it, is unlikely to find much value in unfolded clusters and their parent-child links. Additionally, it is also improbable that an expert would be confused by opened clusternodes disappearing from the screen, since their mental map of the graph is disturbed less when they already have a rough idea of what the result of an operation is going to look like.

While there definitely is further room for improvement the ontology graph shown in the minimap was still integral to many participant's task completion process. With the results presented in chapter 7 it is clear that users perceived it as helpful, a result that is reinforced even further by the fact that every single participant interacted with a cluster present in the minimap at least once. Presenting the full ontology immediately at tool startup enables users to search for specific clusters and to either gain an initial overview of the clusters or reaffirm an already existing mental map.

The interaction transforming nodes according to their position in the ontology graph was only used by 40% of all participants. This is believed to be due to the fact that positions of nodes in the ontology graph were not perceived as meaningful by test subjects. Furthermore, the interaction was generally designed with an expert in mind. No participant attempted to adjust the positioning of nodes within the minimap in order to change the visualization presented after making use of the interaction. Reasons for this lack of use were not explicitly collected over the course of the study to keep it from being too time consuming, but a few were mentioned as comments. For example, the

initial disconnect between node positions in the two graphs was pointed out, as was the fact that it was not always immediately clear to the user which nodes in the ontologygraph were currently visible in the main graph. The most important reason, however, was the fact that the interaction was simply not necessary for the tasks given in the study. While this means that other interactions were sufficient in enabling a user to reach their desired solutions, it also suggests that the interaction is best suited for experts of the *CodeExplorer*. Finally, it is also important to point out that the interaction's use heavily depends on the quality of the imposed ontology. If clusters present little to no meaning to the user, it is unlikely that their positions in the ontology graph will hold any value for them.

Another notable result was the fact that most participants had an idea regarding what is being represented by node size that was similar to the metric used. However, multiple test subjects indicated that they had not recognized a difference in sizes of nodes prior to being asked about it in the questionnaire. This suggests that while node size is a fitting visual attribute to represent the number of classes included in a clusternode, the difference in sizes between nodes was not large enough to emphasize the information given by the metric. On the other hand node sizes can not be increased too much as they might take up excessive amounts of screen space.

A further issue raised by the study's results is the fact that without an explanation of the two different types of clusternodes users were not clear on the fact that a node connected to an aggregatenode was also connected to other clusters hidden within. This led to most participants only indicating the initially visible aggregatenode as a cluster that a node was linked to, instead of unfolding the node and examining its contents. However, it should also be noted that the introduction to the tool given in the first part of the presented study was not as extensive as such an introduction should generally be. Especially information like the fact that an aggregatenode coloured in red contains further clusters, at least one of which is connected to a node that is also connected to the parent, is detrimental to a correct answer to the posed questions, but was intentionally omitted in order to allow for an examination of user's perceptions of the meaning behind the colouring.

Another aspect that was a cause of confusion to some test subjects, but was also intentionally kept out of the introduction to the tool, was the fact that clusters are not distinct. Since there were instances in both scenarios in which a relevant node was part of two clusters, the study aimed to gauge how much the situations confuse users. Overall, most participants were taken aback when a linked cluster was unfolded and no new dependency appeared in the graph, since the relevant node was already unfolded in a different cluster, but were able to quickly identify the reason. This suggests that the potential confusion caused by such situations is present, but not too impactful on a user and therefore does not outweigh the added flexibility of fuzzy clusters. One test subject expressed that it was the name "cluster" that caused the expectation of distinct groups, thereby implying that a name like "group" would have worked better.

Furthermore, multiple participants mistakenly interpreted parent-child links of clusternodes as further dependencies. While this issue could be prevented with a more thorough introduction to the tool, a more distinct colouring of nodes would also help. A trade-off can be found in a distinct colouring putting more emphasis on the difference between links, and a rather muted colouring, like the one used in the study, attempting to keep visual impressions from overwhelming the user.

As for results concerning the cognitive load experienced by test subjects, answers overall indicated a cognitive load that was neither particularly high nor particularly low. However, it is likely that this load will decrease even further over time a user spends using the tool, according to the results of the corresponding question. This suggestion is also apparent in the fact that task completion times and correctness of answers improved in $\mathrm{S2}$ when compared to $\mathrm{S1}$. Overall, participants' comments reveal that their comfort with the tool increased rapidly over the two scenarios. Therefore, it is expected that the tool will become even easier to use when used for longer periods of time, especially when these periods are spent working with the same software graph.

It should also be noted that it is extremely likely that not only a habituation phase with the *Code-Explorer* in general, but also with the visualized software itself impacts user's comfort with the tool, potentially leading to significantly lower levels of cognitive load. However, it is also noteworthy that this means that results of $\mathrm{S2}$ in the study are also significantly impacted by the learning effect experienced after $\mathrm{S1}$.

One important fact to recognize when evaluating the study in terms of its goal of reducing the cognitive load endured by a user is the fact that without the additions made by this thesis the cognitive load on participants of the study would have been immeasurably high. This is due to the fact that users would have had to find specific nodes in the unclustered visualization shown in figure 8.1 by hovering over each node until they found the node they were looking for. Even though a participant could potentially have found the node, hovering over 687 nodes before finding the relevant one is not practical in any real use case. This means that the task is simply impractical and the cognitive load experienced by a user would have been higher than the highest possible answer on the Paas Scale presented in Paas et al. [34] It would have been impossible to keep all already hovered nodes in mind. Note that there is a search function hinted at in the top right corner of the screenshot, but study participants were disallowed from using it to force them to examine the ontology imposed on the software graph. While this means that the task of finding and evaluating individual nodes is possible by use of the search function, a user would be missing out on the additional information given by the ontology and they would have been unable to gather an accurate mental map of the graphs topology without this thesis' additions.

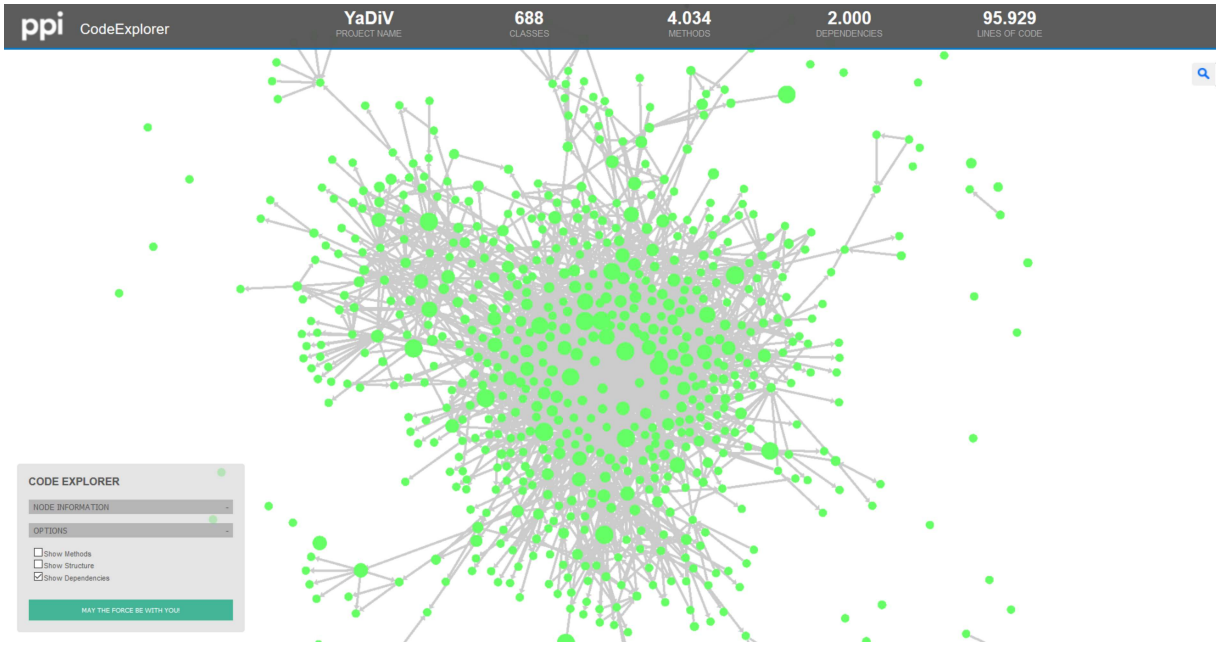


Figure 8.1: The *CodeExplorer* before the work done in this thesis.

All in all, the results found in the evaluation presented in chapter 6 show that the tool is helpful when looking to get an overview of a software code. Every single participant mentioned that one of the main reasons for their rating of the tools usefulness is the fact that the tool makes searches for dependencies and structures within them significantly easier and less tedious than a simple search through individual source code files. This leads to the conclusion that the tool is definitely useful for software engineers beginning to work on a new tool and can also support experienced engineers when attempting to get an overview of their project. It should be noted that these statements are especially meaningful since most of the participants are students of computer science related lines of study who are likely to start working in the field soon. As newly recruited software engineers they will often have to work on a foreign software code which presents one of the use cases this tool was intended for.

Additionally, the goal of this thesis' concepts to decrease the mental effort required to solve tasks using the tool can also be seen as successful, since users could finish tasks they would not have been able to perform without the clusterings. Answers for the question in the questionnaire regarding the experienced cognitive load are also reasonably low, considering that participants were using the tool for the first time while also being introduced to a new software project.

8.2 Limitations

Results found by the study presented in this thesis are very favourable towards a tool like the *CodeExplorer* and the additions made by this thesis. However, there are a number of limitations to the results' validity. The validity threats in this section are ordered by the priority among threats explained by Wohlin et al. [42].

First and foremost for threats to internal validity, one important limitation to recognize is the learning effect that was present in the study and impacted the results of $\mathrm{S2}$. While the seemingly strong learning effect on one hand indicates that the tool can easily be integrated into workflows introducing new hires to a software project, on the other hand it also impacted the results of tasks performed in $\mathrm{S2}$ and therefore the answers regarding the usefulness of the second visualization used. It should be noted though, that the way that the two visualizations were used in the study, namely the variation showing unfolded clusternodes and their parent-child links being used first and the one hiding such nodes only being utilized by more experienced users, fits the design philosophies behind both variations.

A threat to the external validity lies in the fact that there is no baseline to compare the results of the study to. Especially in terms of the cognitive load that this thesis is trying to alleviate, a baseline to compare the answers of participants of the study to would have been useful. However, as described earlier in this chapter, the tasks assigned to test subjects would simply have been impossible without the clustering added by this thesis, since the cognitive load would have been too high to enable practical use of the tool.

Concerning the construct validity it should be noted that a study including actual use cases instead of the theoretical ones featured in this thesis would have been preferable. Due to the ongoing COVID-19 pandemic and data privacy concerns such a study was not possible.

Finally, the conclusion validity is threatened by the sample size of the study being quite small with two participants for the preliminary study and eight participants for the main study. A second, larger experiment would be interesting, even though results are not be expected to differ majorly from the ones found in this work.

Additionally, the steps a user has to take in order to find a fitting answer to a question always depend on the software visualized and the ontology imposed onto it. This means that there is no way to evaluate the tool and the additions made by this thesis without also evaluating the data and especially the ontology visualized. For example, it is much easier to find an entity "car" in a cluster "vehicles" in comparison to an ontology in which the entity belongs to a cluster named "cluster1". Therefore the study's results are also influenced by the data presented.

This means that effects like the confounding effect and the mono-method bias have some impact on the results found. The Rosenthal-effect defined in Rosenthal et al. [36] is also expected to have been present, as participants could ask questions at any point in the study, meaning that experimenters could have impacted a test subject's interaction with the tool subconsciously.

8.3 Expandability

Ideas conceptualized, implemented and evaluated in this thesis generally work for all kinds of graphical structures. Since ontologies present an additional way to include knowledge into the visualization and this knowledge is only limited in the sense that it is used to create groupings on the graph's entities, lots of different kinds of information can be imposed as an ontology. Therefore, the concepts are highly expandable on all graphical structures as long as there exists information that can be used to create meaningful groupings of the nodes in the graph.

One example are communication networks in which nodes represent employees and edges connect employees that communicate with one another regularly. When visualizing the communication of a large corporation in this manner large and highly connected graphs can be created. An idea for an ontology that could improve such a graph would be to group the employees according to the office they work from most of the time. These offices can then be grouped even further based on floor, building, city or even country. Therefore, information like "which office buildings does a specific employee communicate with" or "do the offices in city A and city B communicate with one another" are easily accessible.

Another example could be a tracking of potential infections with COVID-19. By representing people as nodes and having edges connect those individuals who have come in contact with one another, a huge graph can be created. This graph could then be simplified by grouping all individuals infected with the virus into a cluster. This would lead to a visualization that enables a user to more easily differentiate between people who have been in contact with an infected person and others who are very unlikely to be infected, as there is a distance between them and the clusternode grouping all infected individuals. Other possibilities for groupings imposed on such a graph once again are the addresses of persons. For example, information like "people from city X are unlikely to be infected at this time since they have not been in contact with individuals living in cities with large numbers of infections" could easily be found this way. Some of the required data could already be present in the "Corona-Warn-App" available in Germany.

9 Conclusion

9.1 Summary

In this thesis, a tool visualizing software code as a graph consisting of nodes representing individual software classes and edges being drawn between nodes whose classes depend on one another, is worked on. Concepts are developed aiming to use ontologies to group nodes of the software graph with the goal of a reduction of visual clutter on the screen. Additionally, fitting interactions are also implemented and an evaluation of the additions to the tool is conducted by means of a user study.

One of the main focusses of this work is the cognitive load experienced by a user. Therefore, concepts are developed attempting to reduce visual clutter by grouping nodes of the software graph in clusternodes. This grouping allows for such clusternodes to substitute nodes included in them within the software graph, thereby greatly reducing the number of nodes and edges presented. Furthermore, the implementation of fold and unfold operations of clusternodes in the graph took the concept of a mental map into account that needs to be preserved to limit the cognitive load experienced by users.

In addition to the basic concepts a minimap showing the imposed ontology in the bottom right corner of the screen and two different variations of the visualization are developed. These two visualizations differ in the styling of unfolded clusternodes with one variation displaying both the unfolded nodes and links between parents and children present on the screen. The other omits such clusternodes and links. Their advantages and disadvantages are also evaluated in the study.

The study itself is conducted as a think aloud experiment with a total of ten participants split between a preliminary study used to validate the study design and a main study collecting the main results. In the study, test subjects were asked to complete two scenarios which required using the tool, before answering general questions regarding their experience with the tool.

Results of the study were generally favourable for both the tool itself and the additions made by this thesis with all participants expressing that they would recommend the tool for similar use cases like the ones presented in the study.

9.2 Future Work

While the concepts developed and evaluated in this thesis led to favourable results in the study there are still many aspects of the visualization and interaction with the tool that are yet to be explored.

One adjustment mentioned by participants of the study was the idea to include an interaction allowing users to designate a node as a focus point. This focus point would remain specially styled in the graph allowing a user to easily relocate it after selecting other nodes to examine them.

Other ideas not explored in this work concern the minimap presenting the ontology graph. One possible improvement would be a way to interact with the size of the minimap. While the use of such an interaction always depends on the familiarity of a user with the presented ontology and the ontology's quality, a general possibility to in- or decrease the size of the minimap can still be helpful. There could even be further work done to enable a user to adjust the ontology itself within the tool by adding, moving or removing clusternodes. One imaginable scenario would be to combine the two adjustments in a way where the interactions adjusting the ontology itself are only available when the minimap is expanded to a certain threshold size, or alternatively providing a button in the presentation switching between two modes. One mode could be the current visualization while the other uses the full screen space to show the ontology graph, while also presenting the new interaction methods to make adjustments.

Furthermore, changes to the styling of nodes in the minimap could be useful. Multiple participants of the study expressed a desire for a different styling of clusternodes that are currently unfolded in the main graph. Styling nodes in this way allows users to easily figure out the current traversal level of each branch of the ontology graph by simply looking at the minimap. Different styles of nodes would require another evaluation and another study.

In addition the different link types are not assigned weights in the layout algorithm at this stage of development. Edge weights could allow the visualization to adjust its emphasis between focussing on the dependencies of nodes and focussing on the groupings created by the ontology. It would even be possible to include sliders into the visualization that enable the user to alter these weights, and therefore the focus of the presentation, while using the tool.

As for the assignment of tags to specific classes, a different method of input would be quite helpful to the tool's overall usability. Since tags at this point in development have to be assigned to each class in the *XML* file individually, the task is quite lengthy and tedious when working on larger software projects. The project visualized in the study of this thesis for example required an addition of tags to all but one of the 688 classes present.

Another potential way to aid a user with tasks performed using the tool is the use of degree of interest functions to indicate potential nodes of interest in the graph. The degree of interest function itself could be defined based on the ontology to put an even bigger emphasis on the information contained in the groupings. A good implementation of such functions would allow the tool to guide a user to further nodes relevant to their current task.

Participants also indicated that functionality to search for a node in the graph and to undo the most recent action would have been useful for the tasks performed over the course of the study.

While in this thesis an ontology needs to be defined before it can be visualized in the tool, it is also imaginable to begin the visualization of the software graph without any clustering. In such a situation clustering algorithms like *k-means* could be used to recommend groups of nodes to the user as potentially interesting clusters.

Generally speaking, further evaluations are required to validate the results found in this thesis. It would be optimal for such a study to be conducted using real world use cases with users being introduced to a new software project using the tool or migration processes being planned using it.

Bibliography

[1] *OWL Web Ontology Language*. https://www.w3.org/TR/owl-features/, . – Accessed: 2020-07-24

[2] *RDF Schema 1.1*. https://www.w3.org/TR/rdf-schema/, . – Accessed: 2012-07-24

[3] *International Conference on Information Visualisation, IV 2002, London, England, UK, July 10-12, 2002*. IEEE Computer Society . – ISBN 0–7695–1656–4

[4] Amar, R. A. ; Eagan, J. ; Stasko, J. T.: Low-Level Components of Analytic Activity in Information Visualization. In: Stasko, J. T. (Hrsg.) ; Ward, M. O. (Hrsg.): *IEEE Symposium on Information Visualization (InfoVis 2005), 23-25 October 2005, Minneapolis, MN, USA*, IEEE Computer Society, 111–117

[5] Amis, E. S.: Coulomb's law and the quantitative interpretation of reaction rates. In: *Journal of Chemical Education* 29 (1952), Nr. 7, S. 337

[6] Antwerp, M. V. ; Madey, G. R.: The Importance of Social Network Structure in the Open Source Software Developer Community. In: *43rd Hawaii International International Conference on Systems Science (HICSS-43 2010), Proceedings, 5-8 January 2010, Koloa, Kauai, HI, USA*, IEEE Computer Society, 1–10

[7] Archambault, D. W. ; Munzner, T. ; Auber, D. : GrouseFlocks: Steerable Exploration of Graph Hierarchy Space. In: *IEEE Trans. Vis. Comput. Graph.* 14 (2008), Nr. 4, 900–913. http://dx.doi.org/10.1109/TVCG.2008.34. – DOI 10.1109/TVCG.2008.34

[8] Bikakis, N. ; Sellis, T. K.: Exploration and Visualization in the Web of Big Linked Data: A Survey of the State of the Art. In: *CoRR* abs/1601.08059 (2016). http://arxiv.org/abs/1601.08059

[9] Corcho, Ó. ; Fernández-López, M. ; Gómez-Pérez, A. : Ontological Engineering: Principles, Methods, Tools and Languages. Version: 2006. https://doi.org/10.1007/3-540-34518-3_1. In: Calero, C. (Hrsg.) ; Ruiz, F. (Hrsg.) ; Piattini, M. (Hrsg.): *Ontologies for Software Engineering and Software Technology*. Springer, 1–48

[10] Cui, W. ; Zhou, H. ; Qu, H. ; Wong, P. C. ; Li, X. : Geometry-Based Edge Clustering for Graph Visualization. In: *IEEE Trans. Vis. Comput. Graph.* 14 (2008), Nr. 6, 1277–1284. http://dx.doi.org/10.1109/TVCG.2008.135. – DOI 10.1109/TVCG.2008.135

[11] Dudás, M. ; Lohmann, S. ; Svátek, V. ; Pavlov, D. : Ontology visualization methods and tools: a survey of the state of the art. In: *Knowledge Eng. Review* 33 (2018), e10. http://dx.doi.org/10.1017/S0269888918000073. – DOI 10.1017/S0269888918000073

[12] Elmqvist, N. ; Fekete, J. : Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. In: *IEEE Trans. Vis. Comput. Graph.* 16 (2010), Nr. 3, 439–454. http://dx.doi.org/10.1109/TVCG.2009.84. – DOI 10.1109/TVCG.2009.84

[13] Ersoy, O. ; Hurter, C. ; Paulovich, F. V. ; Cantareiro, G. ; Telea, A. : Skeleton-Based Edge Bundling for Graph Visualization. In: *IEEE Trans. Vis. Comput. Graph.* 17 (2011), Nr. 12, 2364–2373. http://dx.doi.org/10.1109/TVCG.2011.233. – DOI 10.1109/TVCG.2011.233

[14] Euler, L. : LEONHARD EULER AND THE KOENIGSBERG BRIDGES. In: *Scientific American* 189 (1953), Nr. 1, 66–72. http://www.jstor.org/stable/24944279. – ISSN 00368733, 19467087

[15] Frishman, Y. ; Tal, A. : Dynamic Drawing of Clustered Graphs. In: Ward, M. O. (Hrsg.) ; Munzner, T. (Hrsg.): *10th IEEE Symposium on Information Visualization (InfoVis 2004), 10-12 October 2004, Austin, TX, USA*, IEEE Computer Society, 191–198

[16] Fruchterman, T. M. J. ; Reingold, E. M.: Graph Drawing by Force-directed Placement. In: *Softw., Pract. Exper.* 21 (1991), Nr. 11, 1129–1164. http://dx.doi.org/10.1002/spe.4380211102. – DOI 10.1002/spe.4380211102

[17] Fu, B. ; Noy, N. F. ; Storey, M. D.: Eye tracking the user experience - An evaluation of ontology visualization techniques. In: *Semantic Web* 8 (2017), Nr. 1, 23–41. http://dx.doi.org/10.3233/SW-140163. – DOI 10.3233/SW–140163

[18] Gansner, E. R. ; North, S. C.: An open graph visualization system and its applications to software engineering. In: *Software: practice and experience* 30 (2000), Nr. 11, S. 1203–1233

[19] Gershon, N. D. ; Card, S. K. ; Eick, S. G.: Information visualization tutorial. In: Atwood, M. E. (Hrsg.): *CHI '99 Extended Abstracts on Human Factors in Computing Systems, CHI Extended Abstracts '99, Pittsburgh, Pennsylvania, USA, May 15-20, 1999*, ACM, 149–150

[20] Holten, D. ; Wijk, J. J.: Force-Directed Edge Bundling for Graph Visualization. In: *Comput. Graph. Forum* 28 (2009), Nr. 3, 983–990. http://dx.doi.org/10.1111/j.1467-8659.2009.01450.x. – DOI 10.1111/j.1467–8659.2009.01450.x

[21] Horwitz, S. ; Reps, T. W.: The Use of Program Dependence Graphs in Software Engineering. In: Montgomery, T. (Hrsg.) ; Clarke, L. A. (Hrsg.) ; Ghezzi, C. (Hrsg.): *Proceedings of the 14th International Conference on Software Engineering, Melbourne, Australia, May 11-15, 1992*, ACM Press, 392–411

[22] Huang, W. ; Eades, P. ; Hong, S. : Measuring effectiveness of graph visualizations: A

cognitive load perspective. In: *Information Visualization* 8 (2009), Nr. 3, 139–152. http://dx.doi.org/10.1057/ivs.2009.10. – DOI 10.1057/ivs.2009.10

[23] Joblin, M. ; Apel, S. ; Hunsen, C. ; Mauerer, W. : Classifying developers into core and peripheral: an empirical study on count and network metrics. In: Uchitel, S. (Hrsg.) ; Orso, A. (Hrsg.) ; Robillard, M. P. (Hrsg.): *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, IEEE / ACM, 164–174

[24] Kiesling, S. ; Klünder, J. ; Fischer, D. ; Schneider, K. ; Fischbach, K. : Applying Social Network Analysis and Centrality Measures to Improve Information Flow Analysis. In: Abrahamsson, P. (Hrsg.) ; Jedlitschka, A. (Hrsg.) ; Nguyen-Duc, A. (Hrsg.) ; Felderer, M. (Hrsg.) ; Amasaki, S. (Hrsg.) ; Mikkonen, T. (Hrsg.): *Product-Focused Software Process Improvement - 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings* Bd. 10027 (Lecture Notes in Computer Science), 379–386

[25] Klünder, J. : *Analyse der Zusammenarbeit in Softwareprojekten mittels Informationsflüssen und Interaktionen in Meetings*, University of Hanover, Hannover, Germany, Diss., 2019. http://d-nb.info/1185200088

[26] Knuth, D. E.: Two thousand years of combinatorics. In: *Combinatorics: Ancient & Modern* (2013), S. 3–37

[27] Kratochvíl, J. (Hrsg.): *Graph Drawing, 7th International Symposium, GD'99, Stirín Castle, Czech Republic, September 1999, Proceedings*. Bd. *1731*. Springer (Lecture Notes in Computer Science)

[28] Landesberger, T. von ; Kuijper, A. ; Schreck, T. ; Kohlhammer, J. ; Wijk, J. J. ; Fekete, J. ; Fellner, D. W.: Visual Analysis of Large Graphs: State-of-the-Art and Future Research Challenges. In: *Comput. Graph. Forum* 30 (2011), Nr. 6, 1719–1749. http://dx.doi.org/10.1111/j.1467-8659.2011.01898.x. – DOI 10.1111/j.1467–8659.2011.01898.x

[29] Lee, B. ; Plaisant, C. ; Parr, C. S. ; Fekete, J. ; Henry, N. : Task taxonomy for graph visualization. In: Bertini, E. (Hrsg.) ; Plaisant, C. (Hrsg.) ; Santucci, G. (Hrsg.): *Proceedings of the 2006 AVI Workshop on BEyond time and errors: novel evaluation methods for information visualization, BELIV 2006, Venice, Italy, May 23, 2006*, ACM Press, 1–5

[30] Lohmann, S. ; Díaz, P. ; Aedo, I. : MUTO: the modular unified tagging ontology. In: Ghidini, C. (Hrsg.) ; Ngomo, A. N. (Hrsg.) ; Lindstaedt, S. N. (Hrsg.) ; Pellegrini, T. (Hrsg.): *Proceedings the 7th International Conference on Semantic Systems, I-SEMANTICS 2011, Graz, Austria, September 7-9, 2011*, ACM (ACM International Conference Proceeding Series), 95–104

[31] Lohmann, S. ; Negru, S. ; Haag, F. ; Ertl, T. : VOWL 2: User-Oriented Visualization of Ontologies. In: Janowicz, K. (Hrsg.) ; Schlobach, S. (Hrsg.) ; Lambrix, P. (Hrsg.) ; Hyvönen, E. (Hrsg.): *Knowledge Engineering and Knowledge Management - 19th International Conference, EKAW 2014, Linköping, Sweden, November 24-28, 2014. Proceedings* Bd. 8876, Springer (Lecture Notes in Computer Science), 266–281

[32] Lohmann, S. ; Negru, S. ; Haag, F. ; Ertl, T. : Visualizing ontologies with VOWL. In: *Semantic Web* 7 (2016), Nr. 4, 399–419. http://dx.doi.org/10.3233/SW-150200. – DOI 10.3233/SW–150200

[33] Nesbitt, K. V. ; Friedrich, C. : Applying Gestalt Principles to Animated Visualizations of Network Data. In: *International Conference on Information Visualisation, IV 2002, London, England, UK, July 10-12, 2002*, IEEE Computer Society, 737–743

[34] Paas, F. G.: Training strategies for attaining transfer of problem-solving skill in statistics: A cognitive-load approach. In: *Journal of educational psychology* 84 (1992), Nr. 4, S. 429

[35] Perer, A. ; Shneiderman, B. : Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis. In: Czerwinski, M. (Hrsg.) ; Lund, A. M. (Hrsg.) ; Tan, D. S. (Hrsg.): *Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI 2008, 2008, Florence, Italy, April 5-10, 2008*, ACM, 265–274

[36] Rosenthal, R. ; Fode, K. L.: The effect of experimenter bias on the performance of the albino rat. In: *Behavioral Science* 8 (1963), Nr. 3, S. 183–189

[37] Rychlewski, J. : On Hooke's law. In: *Journal of Applied Mathematics and Mechanics* 48 (1984), Nr. 3, S. 303–314

[38] Shneiderman, B. : The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In: *Proceedings of the 1996 IEEE Symposium on Visual Languages, Boulder, Colorado, USA, September 3-6, 1996*, IEEE Computer Society, 336–343

[39] Storey, M. D. ; Noy, N. F. ; Musen, M. A. ; Best, C. ; Fergerson, R. W. ; Ernst, N. A.: Jambalaya: an interactive environment for exploring ontologies. In: Hammond, K. J. (Hrsg.) ; Gil, Y. (Hrsg.) ; Leake, D. (Hrsg.): *Proceedings of the 7th International Conference on Intelligent User Interfaces, IUI 2002, San Francisco, California, USA, January 13-16, 2002*, ACM, 239–239

[40] Toosi, F. G. ; Nikolov, N. S. ; Eaton, M. : Simulated Annealing as a Pre-Processing Step for Force-Directed Graph Drawing. In: Friedrich, T. (Hrsg.) ; Neumann, F. (Hrsg.) ; Sutton, A. M. (Hrsg.): *Genetic and Evolutionary Computation Conference, GECCO 2016, Denver, CO, USA, July 20-24, 2016, Companion Material Proceedings*, ACM, 997–1000

[41] Wiens, V. ; Lohmann, S. ; Auer, S. : Semantic Zooming for Ontology Graph Visualizations.

In: Corcho, Ó. (Hrsg.) ; Janowicz, K. (Hrsg.) ; Rizzo, G. (Hrsg.) ; Tiddi, I. (Hrsg.) ; Garijo, D. (Hrsg.): *Proceedings of the Knowledge Capture Conference, K-CAP 2017, Austin, TX, USA, December 4-6, 2017*, ACM, 4:1–4:8

[42] Wohlin, C. ; Runeson, P. ; Höst, M. ; Ohlsson, M. C. ; Regnell, B. : *Experimentation in Software Engineering*. Springer. http://dx.doi.org/10.1007/978-3-642-29044-2. http://dx.doi.org/10.1007/978-3-642-29044-2. – ISBN 978–3–642–29043–5

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 24.07.2020

Lukas Nagel

Appendix

The appendix includes a consent form regarding names and logos used in this thesis. Additionally, all documents used for the different studies outlined in chapter 6 can be found. Note that these documents are in German as every single participant was German.

1. Consent form
2. Questionnaire used for the tag collection interview
3. Form regarding data collection
4. Questionnaire used for the main study
5. A collection of comments made over the course of the study

Interview - Erhebung von Tags

Projekt: YadiV

Teilnehmer-Nr.: 1 Studien-Nr.: 1

A.1 Datum: 18.05.2020 Uhrzeit: 15:00 Uhr

A.2 Wird eine Aufnahme des Gesprächs akzeptiert?

-

A.3 Geschlecht bzw. präferierte Pronomen:

-

A.4 Alter:

-

A.5 In welchem Arbeitsbereich arbeiten Sie?

-

A.6 Inwiefern stimmen Sie der folgenden Aussage zu?

"Ich bin mit der Architektur des im CodeExplorer visualisierten Softwareprojekts (HBCI Programs oder Yadiv) vertraut."

- 1 - stimme gar nicht zu
- 2 - stimme eher nicht zu
- 3 - unentschieden
- 4 - stimme eher zu
- 5 - stimme voll zu

Beispiel für eine Gruppe:

- Level 1: Code Entitäten, die sich mit Datenbankzugriffen befassen
- Level 2: Alle Code Entitäten, über die Daten in die Prozeduren des Projekts einfließen, oder die Daten aus dem Projekt abspeichern (auch I/O)

A.7 Erhebung von Tags

A7.1 Welche Tags würden Sie den verschiedenen Code Entitäten zuteilen?

-

A7.2 Welche Gruppierungen kennen Sie innerhalb der Code Entitäten?

-

A.8 Erhebung der Tagfindungsstrategie

A8.1 Was war Ihre Strategie, um den Code Entitäten passende Tags zuzuweisen?

-

A8.2 Inwiefern stimmen Sie den folgenden Aussagen zu?

“Ich habe mich bei der Einteilung von Tags von den zuvor genannten Gruppierungen leiten lassen.”

- 1 - stimme gar nicht zu
- 2 - stimme eher nicht zu
- 3 - unentschieden
- 4 - stimme eher zu
- 5 - stimme voll zu

“Ich habe bei der Einteilung von Tags auf bereits in der Visualisierung des CodeExplorers entstandene Gruppen geachtet.”

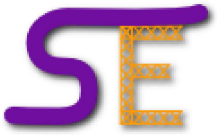
- 1 - stimme gar nicht zu
- 2 - stimme eher nicht zu
- 3 - unentschieden
- 4 - stimme eher zu
- 5 - stimme voll zu

“Die von mir während dieses Interviews vorgeschlagenen Tags sind aussagekräftig.”

- 1 - stimme gar nicht zu
- 2 - stimme eher nicht zu
- 3 - unentschieden
- 4 - stimme eher zu
- 5 - stimme voll zu

A.9 Gibt es noch Ergänzungen oder Punkte, die während dieses Interviews noch nicht zur Geltung kamen?

-



Teilnahme an einer wissenschaftlichen Studie zur Analyse von Ontologiebasierten Visualisierungen für Software

Durchgeführt von Lukas Nagel im Rahmen einer Masterarbeit am Fachgebiet Software Engineering, Leibniz Universität Hannover

Bitte lesen Sie sich dieses Dokument sorgfältig durch. Es dient dazu, Ihnen die Studie vorzustellen und Sie auf Ihre Rechte als freiwilliger Proband / freiwillige Probandin hinzuweisen. Bei Fragen oder Unklarheiten stehe ich Ihnen gerne zur Verfügung.

Vielen Dank für Ihr Interesse an unserer Forschung und Ihre Bereitschaft, an unserer Studie teilzunehmen. Ihre Teilnahme hilft uns, das Konzept zur Visualisierung von Softwarecode zu validieren und zu evaluieren, d.h., es im Hinblick auf die Anwendbarkeit und Güte der Ergebnisse zu untersuchen.

Erhobene Daten

Unsere Studie erhebt mehrere Daten in Form von Antworten auf einen Fragebogen. Teile davon sind Demographische Daten, welche nicht einer Zuordnung, sondern einer Einordnung der Allgemeingültigkeit der Ergebnisse dienen. Außerdem würden wir den Ablauf der Studie gerne über eine Bildschirmaufnahme und ein Mikrofon aufzeichnen, um im Nachhinein jegliche Hinweise Ihrerseits in die Ergebnisse mit einbeziehen zu können.

Datenschutz und Datenspeicherung

Die von Ihnen zur Verfügung gestellten Daten werden ausschließlich anonym und ohne Rückschlüsse auf einzelne Personen ausgewertet:
Vor der Verarbeitung ihrer erfolgt eine umfangreiche Anonymisierung. Die Anonymisierung kann wahlweise von Ihnen vor Aushändigung der Daten oder von einem Mitarbeiter oder einer Mitarbeiterin des Fachgebiets Software Engineering nach Unterzeichnung einer Verschwiegenheitserklärung durchgeführt werden.
Zum jetzigen Zeitpunkt ist noch eine Publizierung der Ergebnisse möglich. Diese geschieht natürlich ausschließlich mit den anonymisierten Datensätzen. Dafür ist eine Datenspeicherung auf unserem Server für einen begrenzten Zeitraum erforderlich. Die Datenspeicherung erfolgt erst nach der Anonymisierung.

Ablauf und Dauer der Studie

Die Studie besteht aus einem umfassenden Fragebogen, inklusive demographischen Fragen, Aufgabenstellungen zum Tool und Fragen zur Benutzbarkeit, sowie einer Einführung in die Verwendung des Tools. Wir rechnen mit einer Studiendauer von unter 60 Minuten pro Teilnehmer.

Abbruch der Studie

Sie haben zu jedem Zeitpunkt die Möglichkeit, die Teilnahme an der Studie abzubrechen. In diesem Fall werden Ihre gesamten Daten und die Auswertungsergebnisse gelöscht.

Einverständniserklärung zur Teilnahme an der wissenschaftlichen Studie zur Analyse von Stimmung in Teams anhand von schriftlicher Kommunikation

Diese Studie wird im Rahmen der Masterarbeit „An Ontology-Based Approach to Visualize Large Software Graphs“ durchgeführt. Damit Sie an der Studie teilnehmen können, bestätigen Sie uns bitte folgende Aussagen:

Durchführung der Studie

Ich wurde über den Ablauf der Studie, die erhobenen Daten und die Datenverarbeitung informiert.

Datenaufzeichnung

Ich wurde darüber informiert, dass für die Studie Bildschirm und Ton aufgezeichnet werden. Dazu werden die Daten für die Dauer der Studie auf Servern der Leibniz Universität Hannover in anonymisierter Form gespeichert. Die aufgezeichneten Daten werden nur für die wissenschaftliche Forschung genutzt und ausschließlich anonymisiert ausgewertet.

Mit der Veröffentlichung der vollständig anonymisierten Ergebnisse in wissenschaftlichen Publikationen bin ich

- □ einverstanden.
- □ nicht einverstanden.

Ich habe zur Kenntnis genommen, dass ich ein Recht darauf habe, jederzeit Auskünfte über die gespeicherten Daten zu erhalten. Ich kann die Studie zu jedem Zeitpunkt durch Widerruf dieser Einverständniserklärung abbrechen. Nach erfolgtem Widerruf werden alle Kommunikationsdaten, die in Zusammenhang mit mir stehen (d.h. insbesondere die Kommunikationsdaten des gesamten Teams), gelöscht und für weitere Analysen nicht mehr verwendet.

Mit den aufgeführten Punkten bin ich

- □ einverstanden.
- □ nicht einverstanden.

Ich habe den Überblick über die Studie gelesen und verstanden. Ich bin mit den aufgeführten Punkten in dieser Einverständniserklärung einverstanden. Ich nehme freiwillig und ohne Vergütung an der Studie teil, und bin dazu auch gesundheitlich in der Lage. Ich habe das Recht, die Teilnahme jederzeit und ohne Angabe von Gründen abzubrechen.

_____

Nachname, Vorname (wird zur Sicherstellung der vollständigen Löschung benötigt)

_____

Ort, Datum, Unterschrift

Fragebogen

Teilnehmer-Nr.: Studien-Nr.:

Datum: Uhrzeit: -

A.1 Wird eine Aufnahme des Gesprächs akzeptiert?

- ☐ Ja
- ☐ Nein

A.2 Geschlecht bzw. präferierte Pronomen:

_____

A.3 Alter:

_____

A.4 In welchem Arbeitsbereich arbeiten bzw. studieren Sie?

_____

A.5 Welche Vorerfahrungen haben Sie mit dem Tool “YaDiV” und dem dahinterstehenden Code?

_____

_____

Bevor Sie umblättern haben Sie 5 Minuten Zeit, um sich mit dem Tool vertraut zu machen.

Sie sind neues Mitglied in einem Team von Softwareentwicklern, die am Tool *YaDiV* arbeiten. Ihre erste Aufgabe ist es, eine Klasse *Interpolation* anzupassen, die laut ihren Kollegen zum Cluster *UtilCluster* gehört. Zur Bewältigung dieser Aufgabe wird Ihnen das Tool *CodeExplorer* zur Verfügung gestellt, mit dem Sie sich einen Überblick über das Projekt und vor allem die nötigen Abhängigkeiten der von Ihnen zu verändernden Klasse verschaffen sollen.

A.6 Welche Cluster müssen geöffnet werden, um die Klasse anzeigen zu lassen?

_____

_____

_____

_____

_____

A.7 Welche Cluster haben Abhängigkeiten zu dieser Klasse?

_____

_____

_____

_____

_____

A.8 Welche Klassen innerhalb dieser Cluster sind besonders interessant?

_____

_____

_____

_____

_____

Ab hier wird eine andere Variante der Visualisierung verwendet.

Als zweite Aufgabe sollen Sie eine Klasse entwickeln, die ähnlich zu bestehenden Klassen im Cluster MessagesCluster funktioniert. Erneut sollen Sie den “CodeExplorer” zu Rate ziehen.

A.9 Welche Klassen halten Sie für diese Aufgabe für relevant?

_____

_____

_____

_____

_____

A.10 Welche Klassen sind die Schlüsselklassen des Clusters?

_____

_____

_____

_____

_____

A.11 Zu welchen Clustern und Klassen hat die Klasse “YObserver” Abhängigkeiten?

Cluster:

_____

_____

Klassen:

_____

_____

_____

Nun sollen einige Frage zu Ihrer Erfahrung mit dem “CodeExplorer” beantwortet werden.

A.12 Inwiefern stimmen Sie den folgenden Aussagen zu?

A.12-1 *“Die Darstellung mit versteckten Clusterknoten und -kanten nach Aufklappen des Knotens ist verständlicher als die andere Variante.”*

- ▢ 1 - stimme gar nicht zu
- ▢ 2 - stimme eher nicht zu
- ▢ 3 - unentschieden
- ▢ 4 - stimme eher zu
- ▢ 5 - stimme voll zu

A.12-2 *“Die Minimap in der unteren rechten Ecke war hilfreich.”*

- ▢ 0 - wurde nicht verwendet
- ▢ 1 - stimme gar nicht zu
- ▢ 2 - stimme eher nicht zu
- ▢ 3 - unentschieden
- ▢ 4 - stimme eher zu
- ▢ 5 - stimme voll zu

A.12-3 *“Die durch die Taste T ausgeführte Interaktion war hilfreich.”*

- ▢ 0 - wurde nicht verwendet
- ▢ 1 - stimme gar nicht zu
- ▢ 2 - stimme eher nicht zu
- ▢ 3 - unentschieden
- ▢ 4 - stimme eher zu
- ▢ 5 - stimme voll zu

A.12-4 *“Die durch die Taste T ausgeführte Interaktion braucht eine Gewöhnungsphase.”*

- ▢ 0 - wurde nicht verwendet
- ▢ 1 - stimme gar nicht zu
- ▢ 2 - stimme eher nicht zu
- ▢ 3 - unentschieden
- ▢ 4 - stimme eher zu
- ▢ 5 - stimme voll zu

A.13 Was wird durch die Größe der Knoten repräsentiert?

_____

_____

A.14 Was unterscheidet die verschieden gefärbten Knoten?

_____

_____

_____

A.15 Wie groß war die mentale Anstrengung, die Sie während der Verwendung des Tools erfahren haben?

- ▢ 1 sehr sehr geringe mentale Anstrengung (Fahrrad fahren)
- ▢ 2
- ▢ 3
- ▢ 4
- ▢ 5 weder hohe noch geringe mentale Anstrengung
- ▢ 6
- ▢ 7
- ▢ 8
- ▢ 9 sehr sehr hohe mentale Anstrengung (Schreiben einer Klausur)

A.16 Inwiefern stimmen Sie der folgenden Aussage zu?

"Die mentale Anstrengung, die ein Nutzer während der Verwendung des Tools erfährt, wird nach mehrfacher Verwendung des Tools bedeutend geringer sein."

- ▢ 1 - stimme gar nicht zu
- ▢ 2 - stimme eher nicht zu
- ▢ 3 - unentschieden
- ▢ 4 - stimme eher zu
- ▢ 5 - stimme voll zu

A.17 Was hätten Sie außerdem gerne in der Darstellung gesehen?

_____

_____

_____

_____

A.18 Haben Sie allgemeine Anmerkungen oder Vorschläge für die Visualisierung des Tools?

_____

_____

_____

_____

A.19 Inwiefern stimmen Sie der folgenden Aussage zu?

“Die Verwendung des Tools kann die Einarbeitung in ein Projekt unterstützen."

- ▢ 1 - stimme gar nicht zu
- ▢ 2 - stimme eher nicht zu
- ▢ 3 - unentschieden
- ▢ 4 - stimme eher zu
- ▢ 5 - stimme voll zu

A.20 Bitte erklären Sie Ihre Bewertung.

_____

_____

_____

_____

_____

Vielen Dank für Ihre Teilnahme an meiner Studie!

Interaktionen

Doppelclick: Klappt entfaltete Clusterknoten wieder ein oder entfaltet eingeklappte Clusterknoten.

Taste “E”: Klappt den im Hauptgraphen ausgewählten Clusterknoten ein, falls er entfaltet ist, oder entfaltet ihn.

Taste “R”: Klappt den im Ontologiegraphen ausgewählten Clusterknoten im Hauptgraphen ein, falls er entfaltet ist, oder entfaltet ihn.

Taste “T”: Transformiert die Positionen der im Hauptgraphen enthaltenen Clusterknoten anhand ihrer Positionen im Ontologiegraphen.

Taste “P”: Pinnt einen Knoten, sodass er sich nicht mehr von der Stelle bewegt.

Reset: Sie können die Darstellung zurücksetzen, indem Sie die Seite im Browser aktualisieren und im neu auftauchenden Fenster “Show Structure” ab- und “Show Dependencies” anwählen.

Achtung: Die Suchfunktion in der oberen rechten Ecke soll explizit NICHT verwendet werden!

Comments Made During The Study

The following table contains all comments made by participants regarding the *CodeExplorer*[2]. Comments that were made twice are omitted. The letter in front of a number represents whether the comment concerns functionality (F), the visualization (V), or is just a general comment (C).

Number	Comment
F.1	A search function would have been useful for both the main graph and the minimap.
F.2	I would have liked an addition of a legend that appears on screen when a button is pressed.
F.3	The size of the minimap should be adjustable.
F.4	An option to undo the last action would be nice.
F.5	I'd like to be able to mark pinned nodes to create a focus point.
F.6	Pressing the space bar should freeze the renderer.
F.7	An interaction showing which classes in a cluster cause their parent to have a certain dependency link would be helpful.
F.8	I would like to be able to adjust the weights of the different edge types while using the *CodeExplorer*.
F.9	Removing individual dependencies or clusters from the visualization could be useful.
F.10	The mouse cursor should switch its looks based on what part of the visualization it currently hovers over. For example a "click" finger when hovering over nodes or a "pan" cross when hovering over the canvas.
F.11	Hitboxes of nodes should increase in size when zooming out quite far.
F.12	Nodes in a cluster should be highlighted, when their unfolded parent is selected.
F.13	I would like a separate window for large *JavaDoc* files.
F.14	It would be nice if the *CodeExplorer* had a button to directly open the source file of a selected node.
F.15	Clusters should be editable within the minimap.
F.16	There should be a button to show or hide the labels of all base nodes currently presented in the graph.

[2]As all participants of the study were German, comments made during the study were also made in German. However, since this thesis is written in English, comments are translated.

Number	Comment
V.1	Labels should not intersect or be drawn over nodes.
V.2	Padding of labels should scale with the zoom level.
V.3	Clusternodes corresponding to selected nodes should be highlighted in the minimap or main graph.
V.4	The minimap should show which nodes are unfolded and which are not.
V.5	The tool needs better antialiasing.
V.6	Some labels in the ontology are only readable when zoomed in.
V.7	Font colour should change based on the colour of the background.
V.8	The parsing of *JavaDoc* for the information box could be improved.
V.9	I would like to switch between the two visualization variations using a button press.
V.10	The parent of a selected node should also be highlighted.
V.11	Nodes in a cluster should be highlighted, when their unfolded parent is selected.
V.12	The size of base nodes in the graph should scale with their lines of code metric.
C.1	I expected clusters to be distinct from one another.
C.2	All keys used for functionality should be reachable with one hand.
C.3	There is too much movement in the visualization, when the renderer is not frozen.