Finding Structures in Multi-Type Code Couplings with Node-Link and Matrix Visualizations

Ala Abuthawabeh∗, Fabian Beck†, Dirk Zeckzer∗ and Stephan Diehl‡
∗TU Kaiserslautern, Germany
Email: {abuthawa, zeckzer}@informatik.uni-kl.de
†VISUS, University of Stuttgart, Germany
Email: fabian.beck@visus.uni-stuttgart.de
‡University of Trier, Germany
Email: diehl@uni-trier.de

Abstract—Software systems are often modeled and visualized as graphs in order to understand their higher-level structure: code entities are connected by dependencies or couplings. However, when only considering one type of code coupling such as method calls, the understanding gained stays limited to this specific aspect. Encoding multiple types of code coupling in the graph promises to broaden the understanding. Different approaches already exist for visually discerning those types in graph diagrams. In this paper, we study two of these techniques—a node-link and a matrix approach—in a realistic scenario where the classes and interfaces of a system are connected by six different types of code coupling. The explorative user study that we conducted with interactive versions of the two visualizations focuses on getting an insight on how software developers use the visualizations for understanding an unknown system. We classified typical visual structures that the participants were able to identify and connected these structures to software engineering problems. Despite the fundamental difference in approach, the participants identified the same graph structures targeting similar tasks with both visualizations.

I. INTRODUCTION

Directed and undirected graphs are widely used to model and to understand the relations between entities that together form a software system. In software systems, different types of couplings exist between classes and interfaces such as inheritance, aggregation, usage, code clones, evolutionary (co-change) coupling, or semantic similarity. These couplings are complementary to each other, revealing multiple aspects of the same software system. Hence, approaches for analyzing graphs benefit from discerning instead of aggregating different types of edges.

Much research has concentrated on using visual approaches for analyzing graphs in a scalable way [1], but, only recently, some approaches focused specifically on discerning and comparing multiple types of edges [2], [3], [4], [5], [6]. In this work, we selected two complementary approaches and evaluated them in a software engineering scenario. The first approach, which we call parallel node-link visualization (PNLV) [2], depicts multiple types of edges in separate, parallel node-link diagrams that are placed side by side. The second approach, which we call interactive multi-matrix visualization (IMMV) [6], [7], uses an adjacency matrix for visually representing the graph with multiple types of edges in each cell. To our knowledge, these are the only approaches that provide the required scalability (with respect to the number of nodes, edges, and types of edges) to represent coupling graphs of real-world software systems having several types of coupling. Figure 1 contrasts the two approaches used, visualizing the JFtp project.

The specific research question that we want to answer is: “What higher-level coupling structures users are able to retrieve and compare with the help of these visualizations?” Further, we investigate how the detected and compared structures can be used for tasks in the context of software engineering. The main contributions of the paper are the following:

Fig. 1. JFtp project with multiple types of couplings visualized by the parallel node-link visualization (top, detail) and the interactive multi-matrix visualization (bottom, detail).
We evaluate the utility of two recent visual approaches [2], [6], [7] for comparing multiple types of edges in a realistic software engineering scenario.

- We identify visual structures and general graph structures that users find with these visualizations.
- We study the strategies that the participants applied for comparing several types of edges.
- We explore software engineering problems that can be addressed with these visualizations.

For graphs with only edges of a single type, researchers have already conducted comparison studies between node-link and matrix graph visualizations based on predetermined, low-level tasks [8], [9], [10]. These studies suggest that matrix visualizations are often more suitable for analyzing larger and denser graphs. Our study extends these evaluations to multi-type edges and to more complex, higher-level tasks. We designed a realistic application scenario and let eight software engineers gain an understanding of previously unknown software systems. Our observations show which specific software engineering problems can be solved with the two visualization approaches. In contrast to our expectations and previous results on low-level tasks, the two approaches showed very similar characteristics though following contrary visualization paradigms.

The remainder of the work is structured as follows: In Section II, the two evaluated interactive visualization approaches are presented in detail. The experimental design of our study is introduced by Section III. Further, Section IV reports the results of the study by analyzing visual structures, their interactive exploration by the participants, and the participants’ feedback. The results are then discussed in a broader context in Section V. Related work on visually discerning multiple types of edges as well as on comparing node-link and matrix visualizations is presented in Section VI. Finally, Section VII concludes the paper.

II. VISUALIZATION APPROACHES

In this study, we target two related visualization approaches that are briefly introduced in the following: the parallel node-link visualization (PNLV, Section II-A) and the interactive multi-matrix visualization (IMMV, Section II-B). As an illustrating example, Figure 2 presents a small sample software system consisting of one package that includes five classes. These classes are connected by two types of code coupling: inheritance and usage. The system is modeled as a graph—while the classes (and interfaces) of the system form vertices, the different types of code couplings are mapped to different types of edges. Moreover, the package structure is used to hierarchically organize the vertices. Figure 2 shows three variants of visualizing this sample data set: Figure 2(a) depicts a standard node-link representation, Figure 2(b) sketches the diagram in PNLV, and Figure 2(c) represents the data set in IMMV.

A. Parallel Node-Link Visualization (PNLV)

In the parallel node-link visualization (PNLV) [2] as shown in Figure 2(b), multiple node-link diagrams are juxtaposed as columns side by side. In each column, the nodes representing the classes and interfaces are placed above each other on a vertical axis. Each node is split and has one port for all its outgoing edges and one port for all its incoming edges; these ports are aligned horizontally. The edges are directed from the outgoing ports on the left to the incoming ports on the right. Each of the juxtaposed diagrams represents a different type of edges.

A layered icicle plot is attached to the left side of the diagram to show the hierarchy of the software system. The horizontal lines in the icicle plot between boxes that represent packages are extended through the whole visualization. The vertical separators between the juxtaposed node-link diagrams repeat the leaf level of cells in the icicle plot. Inside the icicle plot, nodes display labels if possible or on demand when hovered by the mouse. The visualization is adapted to the window size so that all data is always completely visible.

Highlighting by color is used for discerning a set of selected nodes from non-selected ones as well as for marking their outgoing and incoming edges. Edges starting at selected nodes are colored green, those ending at selected nodes are colored...
red, and those starting and ending at selected nodes are colored brownish-green; a light blue color is used for the other edges. A single node is highlighted by clicking on one of its visual representation in the diagram while a set of nodes contained in a package is selected by clicking on the package. The participant can open the source code of a class or interface by double-clicking.

B. Interactive Multi-Matrix Visualization (IMMV)

In the interactive multi-matrix visualization (IMMV) [6], [7] as shown in Figure 2(c), the classes of a software system and the code couplings between them are visualized using an adjacency matrix representation of the underlying graph. Every cell of the matrix is divided into sub-cells, each sub-cell representing a different edge type (code coupling). A colored sub-cell appears in the matrix if there exists an edge of the respective type from the vertex in the current row to the vertex in the current column. To more clearly distinguish the types, a different color is used for each type; a color legend is attached to the matrix.

Analogous to PNLV, IMMV is combined with layered icicle plots to display the package structure of the software system [7]. A copy of the same icicle plot is placed at the left as well as on the top of the matrix. Packages are labeled and the names of classes and interfaces can be retrieved on demand as tooltips. Further, summaries of the code couplings are integrated into the icicle plots aggregating all couplings of a row or column respectively; these summaries are aggregated on package level and displayed in the package representations. If the matrix is larger than the window, the participant can scroll to explore the complete matrix.

A set of classes and interfaces or packages can be selected by clicking for highlighting them in the icicle plots and in the matrix; the rows and columns of all selected entities become surrounded by a strong black border line. It is further possible to select a cell of the matrix—both respective row and column are highlighted. Again, the participant can open the source code of a class by double-clicking.

III. EXPERIMENT DESIGN

While carefully designing the experiment, we discussed quantitative as well as qualitative approaches but finally decided to conduct a mostly qualitative study in order to explore how users—provided with a very general task—work naturally with the complex visualization approaches. Ellis and Dix [11] argue that explorative studies such as ours are most suitable for understanding the utility of information visualizations. Lam et al. [12] surveyed and classified empirical research in information visualization; with respect to their taxonomy of seven scenarios, our approach falls into the category of evaluating visual data analysis and reasoning (VDAR)—field studies and partially controlled experiments are typical and appropriate for this kind of research investigating how visualization supports data exploration and knowledge discovery.

A. Research Goal

The specific research question that we want to answer is: “What higher-level coupling structures users are able to retrieve and compare with the help of these visualizations?” Hence, the core purpose of our study is to determine which visual structures the participants are able to identify and to interpret in the two presented visualizations in a realistic software engineering context. We define a visual structure as any set of visual elements that are perceptually grouped. On a basic level, perceptual grouping is described by the Gestalt Laws [13]. A visual structure can be, for example, a fan of links (PNLV), a group of equally colored cells (IMMV), or the co-occurrence of types of edges as links in different columns (PNLV) or as colored sub-cells (IMMV).

Working with the visualizations, identifying structures like these is a first step towards making sense of the presented information. As a next step, the visual structures need to be connected to specific graph structures such as vertices having a high degree or strongly coupled clusters of vertices. These graph structures can be embedded into the domain-specific context to finally draw conclusions from the visualization. Our study intends to cover this process of interpretation as summarized in Figure 3 as a whole.

![Fig. 3. Interpretation process for deriving domain-specific insights from a graph visualization.](image)

Finally, we analyzed which strategies the participants applied for comparing different edge types.

B. Experiment Setup

The study was performed as a lab experiment following a counterbalanced, within-subject design: each participant worked with both visualizations. Two people were responsible for managing the experiment: a moderator explaining and leading the experiment and an observer taking notes. The visualizations were first shown to the participants printed in color on A2 paper. In the course of the experiment, the participants also used the interactive versions of the visualizations on a Windows 7 PC with a 23” LCD screen with full HD resolution, a mouse, and a keyboard. The source code files of the data sets were displayed in Notepad++ [14] on demand. Screen and voice were recorded with Camtasia Studio [15].

C. Participants

Eight participants, seven males and one female, volunteered for the experiment. They were between 26 and 45 years old; a color deficiency test showed that they did not have any color vision restrictions. All of them had at least a Master/Diploma degree in Computer Science or Mathematics, as well as programming experience between 3 and 30 years with a
median of 11 years. Seven of them used the Java programming language regularly. The largest sizes of teams they developed software together were between two developers and more than ten developers. Two of them regularly worked with visual representations for software development such as VisDB or UML diagrams. Four participants were professional software developers from industry, the other four from academia.

D. Datasets

Adopted from a study on code coupling [16], six different types of couplings were considered:

- **inheritance**: extend and implement dependencies
- **aggregation**: usage of another class as the type of a class attribute
- **usage**: structural dependencies on method level
- **evolutionary coupling**: files that were changed together frequently in the past
- **code clones**: files that are reasonably covered by the same code clones
- **semantic similarity**: similar vocabulary used in identifiers and comments

These couplings were retrieved for a set of open source Java projects. Table I provides statistics on the sizes of the projects that the participants analyzed: JFtp and JUnit represent small projects, while Stripes and Checkstyle represent medium-size ones. The two small and the two medium-size projects were employed together pairwise in the experiment. An additional data set, JHotDraw, was used as a sample for tutorials and trials.

<table>
<thead>
<tr>
<th></th>
<th>JFtp</th>
<th>JUnit</th>
<th>Stripes</th>
<th>Check.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>category (size)</strong></td>
<td>small</td>
<td>small</td>
<td>medium</td>
<td>medium</td>
</tr>
<tr>
<td># package</td>
<td>8</td>
<td>26</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td># nodes</td>
<td>78</td>
<td>119</td>
<td>238</td>
<td>261</td>
</tr>
<tr>
<td><strong># edges</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inheritance</td>
<td>40</td>
<td>63</td>
<td>143</td>
<td>207</td>
</tr>
<tr>
<td>aggregation</td>
<td>66</td>
<td>35</td>
<td>131</td>
<td>69</td>
</tr>
<tr>
<td>usage</td>
<td>38</td>
<td>251</td>
<td>614</td>
<td>479</td>
</tr>
<tr>
<td>evolutionary coupl.</td>
<td>380</td>
<td>76</td>
<td>262</td>
<td>586</td>
</tr>
<tr>
<td>code clones</td>
<td>22</td>
<td>40</td>
<td>296</td>
<td>440</td>
</tr>
<tr>
<td>semantic similarity</td>
<td>62</td>
<td>236</td>
<td>462</td>
<td>1014</td>
</tr>
</tbody>
</table>

E. Experiment Protocol

The experiment followed the protocol outlined in Table II, which is a within-subject design with two experimental conditions: using PNLV and using IMMV. To counterbalance for biases such as learning and tiring effects, the order of the conditions was varied systematically across all participants (i.e., four participants began with PNLV and four with IMMV). Moreover, different data sets were used in the two conditions: half of the participants used the two small projects (JFtp and JUnit; four participants) and half of the participants used the two medium-size ones (Checkstyle and Stripes; four participants). The order of the data sets was also systematically varied.

At the beginning of the experiment, the moderator introduced the data sets and the different types of code couplings in a short tutorial using a slide show (average: 4 min). Then, the participants analyzed two different software projects of the same size in two different experimental conditions (PNLV: 30 min; IMMV: 26 min). The software projects and the visualizations were combined in a counter-balanced design, both for the combination of software systems and tools and their order of presentation. Allowing the participants to take as much time as they need for the tasks (like in a realistic scenario) resulted in different average times. Finally, the participants were asked to fill in a questionnaire consisting of general demographic questions and specific questions on the two visualizations (18 min). In total, the experiment took 82 minutes on average, including short breaks for switching, for instance, between paper test and tool test.

Each of the experimental conditions started with a training of the participants. First, a brief oral tutorial (power point slides) was given about the respective visualization technique (4 min). Then, the participants were asked to solve two simple tasks with the interactive visualization to familiarize themselves with the tool (PNLV: 5 min; IMMV: 4 min). The JHotDraw data set was used for the tutorials. After the training, the participant performed the first experimental task as a paper-based test (PNLV: 9 min; IMMV: 8 min): “What interesting visual structures do you find in the visualization? Please mark them in the visualization, give them a number, and orally describe them.” Afterward, we switched to the interactive visualization of the same data set and the participants proceeded with the following task (PNLV: 12 min; IMMV: 10 min): “Select three of the numbered visual structures, which appear to be interesting. For each selected structure, please use
the interactive visualization to explore the structure. Explain your findings orally.”

The execution of the tasks was recorded using Camtasia Studio [15]. Both screen and voice were recorded to allow for a more detailed analysis.

At any time, the participants were allowed to ask questions. If a participant obviously had problems with a task or visualization, the moderator or observer provided help. No hard time limits were enforced in any step. To implement a thinking-aloud approach, the participants were asked and occasionally reminded to orally explain their thoughts and findings. The described experimental protocol was evaluated and improved with two additional test participants before the actual experiment was conducted.

IV. RESULTS

Following the outline of our study also for describing the results, we first present visual structures that were identified in the paper tests (Section IV-A); then, we report how the participants explored these structures in the interactive tool tests (Section IV-B), and finally analyze the answers provided in the questionnaires (Section IV-C).

A. Visual Structures (Paper Test)

The participants were asked to mark visual structures in the printed visualizations. We categorized these structures and found four different repeatedly used categories of structures for each visualization approach that almost all of the obtained visual structures can be classified into. In case of PNLV, these categories of visual structures match categories derived theoretically by Burch et al. [17] for a related visualization technique: fan, beam, cross beam, and gap. For IMMV, we named the four categories—according to their visual appearance—line, diagonal cluster, off-diagonal cluster, and empty area. Connecting these categories of visual structures to graph structures, we observed that they are encoding pairwise the same graph information. While Table III provides an overview of the outlined classification scheme, details on the graph structures and related visual structures are discussed in the following; an example detected by the participants illustrates each structure.

1) High Degree: Vertices in the graph having a high degree of edges.

PNLV: In PNLV, they appear as fan-like structures of links, either on the left side for outgoing edges or, more typically for a software project, on the right side for incoming edges.

IMMV: In IMMV, the same graph structures are reflected in horizontal or vertical lines of equally colored sub-cells. For example, four of four participants who analyzed JFtp (two participants used PNLV, two others IMMV) detected a high in-degree for a small set of entities in the framework package for inheritance as well as aggregation (PNLV: 2/2, IMMV: 2/2). Though located in the same package, a closer investigation revealed that these were different entities with respect to inheritance than with respect to aggregation.

2) Within-Package Edges: Groups of edges connecting vertices of the same package were also observed in both visualizations.

PNLV: In PNLV, the participants marked edges forming a beam or an x-shape where the source and destination classes of these edges are included in the same package.

IMMV: In IMMV, equivalent structures appear as blocks of cells on the main diagonal of the matrix, which have the same or a similar combination of sub-cells. A typical example of a set of within-package edges which was detected by the participants is the tag package in Stripes: many entities are connected by code clone couplings to other entities of the same package (PNLV: 2/2, IMMV: 2/2).
3) **Cross-Package Edges:** If a group of edges does not connect the vertices of the same package but of two different packages in the same direction.

**PNLV:** PNLV shows a beam or an x-shaped structure that crosses the borders of packages.

**IMMV:** In IMMV, these structures can be detected as blocks of cells not located on the main diagonal.

For instance, based on evolutionary coupling, the theories package of JUnit is connected by a number of cross-package edges to one of the runners packages (PNLV: 2/2, IMMV: 0/2).

4) **No Edges:** Vertices that do not have any outgoing or incoming edges (to all or a subset of other vertices) form another graph structure.

**PNLV:** In PNLV, these vertices are represented as nodes on the left or right side of a diagram that do not have any links attached (in a specific range) and form kinds of gaps in the linear list of nodes.

**IMMV:** In IMMV, empty areas that do not contain any colored cells (in general or with respect to a specific color) hint at the same graph structure.

Entities without edges can be found frequently, but were only rarely marked by the participants as distinct structures. For instance, one participant detected a larger set of entities within the coding package with PNLV; another participant found with IMMV that the framework package of JFtp lacks within-package edges in general.

Table III also summarizes how many participants identified the respective visual structures in PN LV and in IMMV. Contrasting the two visual structures of each pair, it shows that both were identified by approximately the same number of participants in the two visualizations. A large number of participants identified a vertices with a high degree (PNLV: 7, IMMV: 8), groups of within-package edges (PNLV: 8, IMMV: 8), and groups of cross-package edges (PNLV: 6, IMMV: 7), while only few marked the visual structures classified as no edges (PNLV: 2, IMMV: 2).

Since the visualizations discern between multiple types of edges, the identification of visual structures is also related to visually comparing the different types. In PN LV, these types are viewed side by side; hence, participants needed to mark similar structures across different columns to indicate a comparison: three participants linked fan-in structures, two participants beam structures, and one participant contrasted a beam structure to a gap structure. In contrast, the comparison of types of edges is inherent in IMMV because the different types are shown within the same cells and can neither be perceived nor marked independently of each other. Hence not surprising, more participants identified those multi-type structures in IMMV than in PN LV; in particular, eight participants marked diagonal clusters consisting of multiple types, six participants line structures, and three off-diagonal clusters.

The participants identified the visual structures in a specific sequence. In order to analyze trends, we split the sequence into half (similar to a median split) based on the sequential number that was assigned in the experiments (in cases of an odd number of identified structures, we excluded the one in the middle). Figure 4 reports the results of this analysis split by visualization and graph structure. While we do not observe any fundamental differences, a few trends are notable: vertices with a high degree tend to be more frequently found in the first half than in the second in PN LV while it is the other way around for IMMV; within-package and cross-package structures, in contrast, were found earlier by trend with IMMV than PN LV. These trends might be connected to small differences in difficulties for finding certain structures, to training and experience required, or effects of reading order.

### B. Interactive Exploration (Tool Test)

After analyzing the visualizations on paper, the participants explored three of the identified visual structures using the respective interactive tool. The insights they were able to gain identify specific tasks related to software engineering that can be targeted with the help of the visualizations. We were able to categorize each interactive exploration of a visual structure as one of three software engineering tasks and connected these tasks to analyzed visual structures and applied high-level interaction techniques. Please note that multiple structures or interactions were assigned to a single instance of an interactive exploration (in two cases we were not able to unambiguously assign a visual structure). Moreover, we recorded whether a participant compared different types of couplings during the interactive exploration. Split by the identified software engineering tasks, the results of this classification process are summarized in Table IV and reported in the following.

**Find a central class/interface:** This task, most frequently investigated by the participants (PNLV: 11; IMMV: 9), is closely related to code entities having a high in-degree for structural dependencies such as inheritance, aggregation, and usage. As discussed above and confirmed by Table IV, these can be detected by looking for fan-in structures in PN LV (11 of 11 cases) or vertical lines in IMMV (8 of 9 cases). The interaction strategy typically
Table IV
Software engineering tasks addressed by the participants through the use of the interactive visualization tools; frequency values refer to the number of investigated structures summed for all participants, in square brackets the frequency involving the comparison of multiple types of couplings; structures and interactions are listed if they occurred at least two times, the exact frequency is provided in parentheses.

<table>
<thead>
<tr>
<th>Task</th>
<th>PNLV Structure</th>
<th>Freq.</th>
<th>IMMV Structure</th>
<th>Freq.</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>interaction: select class (11), read source code (7), interpret names (4)</td>
<td></td>
<td>interaction: select class (8), read source code (7), interpret names (4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>understand a package</td>
<td>structure: beam (8), cross beam (2)</td>
<td>10 [8]</td>
<td>structure: diagonal cluster (7), empty area (2)</td>
<td>7 [6]</td>
<td>17 [14]</td>
</tr>
<tr>
<td></td>
<td>interaction: read source code (7), select class (6), select package (5), interpret names (5), compare source code (4)</td>
<td></td>
<td>interaction: interpret the names (5), read source code (4), select package (3), select class (2), compare source code (2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>interaction: read source code (2)</td>
<td></td>
<td>interaction: read source code (5), interpret names (4), select class (3), select class (2), select package (2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Applied was to first select the respective class (PNLV: 11 of 11 cases; IMMV: 8 of 9 cases) and then to open and read the source code of the class (PNLV: 7 of 11 cases; IMMV: 7 of 9 cases); in some instances, interpreting the names of the related classes and interfaces was even already sufficient or made opening the editor superfluous (PNLV: 4 of 11 cases; IMMV: 4 of 9 cases). Depending on the type of coupling, the role of the central class is different: a high inheritance in-degree hints at a central code entity in the inheritance hierarchy while a high usage could identify an important data class. In 11 of the 20 cases, the participants also compared different types of couplings for the selected code entity.

**Understand a package:** Another frequent task that the participants addressed was trying to understand the purpose and characteristics of a particular package (PNLV: 10; IMMV: 7). Often, a set of within-package edges hinted at a particularly interesting package (PNLV: 8 of 10 cases; IMMV: 7 of 7 cases), but also other visual structures occasionally provided a starting point for exploring a package (PNLV: cross beam, IMMV: empty area). The interactions for exploring a package were diverse: in at least half of the cases, the participants read the source code (PNLV and IMMV), interpreted the names of the code entities (PNLV and IMMV), or selected a class or package (PNLV). Sometimes, also source code files were directly compared by quickly switching between the tabs of the editor (PNLV: 4 of 10 cases; IMMV: 2 of 7 cases). Understanding a package typically involved analyzing multiple types of coupling (14 of 17 cases).

**Identify a high-level coupling:** As the least frequent task among the three, the participants analyzed groups of couplings that are similar and together form a kind of high-level coupling, usually between two different packages (PNLV: 2; IMMV 7). Cross-beams (PNLV: 2 of 2 cases) or off-diagonal clusters (IMMV: 4 of 7 cases) usually served as a visual indicator for noteworthy high-level couplings. To further explore these structures, the participants often studied the source code of the connected code entities (PNLV: 2 of 2 cases; IMMV: 5 of 7 cases) and applied other high-level interactions in IMMV. The detected high-level couplings were usually impacted by two or more different types of couplings (8 of 9 cases).

In general, the experiment shows that the participants frequently used the possibility to compare different types of edges: As Figure 5 reports, for 72% of the analyzed structures, the participants compared two or more different types of edges; this also includes structures that were only marked in one type of coupling in the paper test, but extended to multiple types in the interactive test. Contrasting the two visualization approaches again does not reveal any larger difference: the percentage of cases that involved comparing multiple types is similar (PNLV: 74%; IMMV: 70%). This clearly relativizes that we found less visual structures including a comparison of types based on the paper test in PNLV than in IMMV and is remarkable because, as discussed above, it requires an extra effort to do a visual comparison in PNLV.

### C. Questionnaire

As asked for their preference and the usefulness of the visualization, PNLV and IMMV were rated about equal: While four participants preferred IMMV, three liked PNLV more
(one participant could not decide). Moreover, with respect to usefulness, both tools received nearly the same average answers (PNLV: 2.75; IMMV 2.88; scale from strongly agree (1) to strongly disagree (5) that it is useful). Motivating their decision on the preference, the participants provided diverse reasons such as good overview in PNLV and IMMV, familiarity with the node-link paradigm, but also problems with visual clutter in PNLV. For example, one participant stated that “PNLV has good overview with respect to different dependency types and IMMV has good overview with respect to the package structure”. Another participant used PNLV and IMMV on small data sets. He preferred PNLV: “Based on presented examples it seems to be better organized, easier to navigate, or is visually more appealing”. However, using different data set sizes in our experiment decreases such effects that are based on the size.

Regarding software engineering, the participants see the main area of application of the two tools in program comprehension (PNLV: 4 times; IMMV: 4 times) and improving the architecture or design of software systems (PNLV: 4 times, IMMV: 3 times). For PNLV, the participants listed finding central classes as the most interesting insight they gained from the visualization (4 times). For IMMV, the picture is not as clear: participants mentioned different insights, none more than once. When asked if they would like to use the visualizations in their daily software development work, only three participants gave positive feedback for PNLV and one for IMMV. Explaining their reluctance, the participants provided different reasons, among them that the visualizations do not show the right information, that the implementations are not yet ready for practical application, and that the participants themselves are unfamiliar with visualization. Possibly related, only two of the participants declared that they regularly use visual representations for software development.

The participants also had the possibility to propose enhancement for the two tools: For the PNLV tool, the most frequently mentioned missing feature was zooming (4 times)—the original tool actually had a zooming functionality, but it was deactivated for the experiment to make the comparison fair because zooming was not implemented for the IMMV tool. With respect to the IMMV tool, zooming was only mentioned once; improving the selection mechanism and filtering types of edges was proposed by two participants each (the latter was also already implemented but deactivated in the experiment).

V. DISCUSSION

The limitations of our explorative study as well as implications of the results on visualization and on software engineering are discussed in the following.

A. Threats to Validity

By systematically varying the order of the visualizations and data sets, we counterbalanced for possible biases such as learning and tiring effects. We also tried to adjust the two approaches as far as possible by optimizing the readability of both, by deactivating features that were not implemented in one of the approaches, and by using paper versions in parts of the experiment to counterbalance for different interaction techniques. But still, the design or the tool implementations might introduce a bias towards one of the two visualization approaches.

The within-subject design allowed the participants to review and contrast the two visualization approaches in the questionnaire, which provided valuable feedback. But, at the same time, this design biases participants—no matter which visualizations they saw first—to look for similar things in the second visualization. We tried to circumvent parts of the problem by providing different, nevertheless, nearly equally sized, data sets for the two visualizations; but still, the participants might be influenced by this previous experience. The effect could be that the two visualizations showed some more similarities in this study than they would have shown in an equivalent between-subject study.

The quantitative parts of the results have to be interpreted with care because they rely on a small number of participants and are not backed by inference statistics—random effects might have reasonably influenced the numbers. Despite we did not interpret small differences, also our results based on larger difference or similarities in numbers should only be treated as preliminary results that need further quantitative evaluation. Moreover, our study design did not allow for measuring time and accuracy of the analyses the participants performed with the visualizations; the visualization approaches, however, might have shown relevant differences on this level.

Although the transferability of the results is limited by the narrow area of application and the specific data sets, some features of the study also foster transferability: Through considering multiple types of code coupling, the data set includes edge types with different characteristics such as directed and undirected types, or dense and sparse types. For instance, structural dependencies are usually scale-free networks [18]. In contrast, evolutionary couplings usually form denser graphs having many cliques. Moreover, the visual structures relate to general graph structures, hence, are independent of the application domain.

B. Implications on Visualization

Matrix and node-link are two very different metaphors for representing graphs. Also the visual comparison as realized in the two studied visualization approaches is reasonably different: applying the taxonomy of Gleicher et al. [19], PNLV is based on juxtaposition, while IMMV employs a form of superposition (i.e., overlay). But despite those fundamental differences in the visualization approach, both techniques seem to be similarly suitable for investigating multiple types of edges in graphs:

- A comparable number of participants identified the same graph structures (Table III).
- The participants were able to address the same task in the interactive exploration and applied similar interaction strategies (Table IV).
A comparable number of interactively explored structures involved the comparison of multiple types of edges (Figure 5).

In the questionnaire, the participants rated both approaches as equally useful and their personal preference was balanced.

Only small differences are observed between the approaches: In the paper-based test, participants more frequently marked multi-type visual structures in IMMV—maybe due to the superposition approach to visual comparison. In the interactive versions, the selection mechanism seemed to be more important in PNLV. And in the questionnaire, the participants criticized edge clutter in PNLV.

C. Implications on Software Engineering

The identified task that the participants mainly addressed (Table IV) can all be considered as belonging to the application of program comprehension, that is, understanding a software system as required for being able to extend, test, or maintain the system. This matches with the opinion of the participants captured in the questionnaire. Some participants propose also to use the visualizations for improving the architecture or design of systems—this application is not directly reflected in the recorded interactive usage scenarios and will only be possible to capture if the participants already know the analyzed system in detail before the experiment.

The visual distinction of multiple types of couplings seems to add value to solving these tasks: First, similar visual structures for different types of edges refer to different software engineering concepts. Second, the participants frequently applied a visual comparison between two or more types and gained interesting insights from these comparisons—this appears to be particularly important for understanding packages and identifying high-level dependencies.

The connection to the code, that is, the raw data in this experiment, turned out to be very important and was frequently used: while the visualization served as an instrument to navigate, to raise hypotheses, and to answer simple questions, the source code needed to be studied to gain a deeper knowledge about the system, to understand certain couplings, to check the hypotheses, and to answer more complex questions. One participant opened several source documents in Notepad++ [14] to compare these documents. A mechanism for directly opening additional documents in adjacent views would be beneficial for supporting these comparisons. This holds, e.g., for source code related to the different classes connected by an undirected relation (e.g., code clone).

VI. RELATED WORK

The scalable visualization of graphs is an established area of research [1]: within this area, some approaches—like the studied PNLV and IMMV—particularly focus on visually discerning multiple types of edges: Based on node-link diagrams, for instance, Erdemir et al. [4] provided a visualization that uses visual attributes of the links (e.g., color, style, strength) to reflect different attributes of edges; when there exist multiple types of edges between two nodes, only the link having the highest priority is shown. Pretorius and van Wijk [20] introduce special nodes for representing different types of edges—a link goes from the source node through the edge-type node to the target node. Also related are node-link-based graph comparison visualizations where two or more graphs are juxtaposed [21], stacked [22], or contrasted as a visual diff [23]. In a matrix-based representation, an alternative to splitting the cells is only using colors for discerning different types of edges [5], [24], [25], which, however, only scales to a very limited number of edge types.

Related to discerning multiple types of edges, dynamic graphs discern multiple points in time for vertices as well as for edges. Hence, visualizing dynamic graphs is similar to visually comparing different types of edges. Animated node-link diagrams [26], [27], the standard approach to dynamic graph visualization, however, only allow for comparing edges in directly consecutive time steps. In contrast, recent timeline-based approaches support the comparison of edges across larger time spans: For instance, the Parallel Edge Splitting technique [17] uses juxtaposed node-link diagrams and can be considered as a variant of PNLV. But also matrices are used in this context such as the Pixel Oriented Matrix [28], which splits matrix cells into sub-cells similar to IMMV.

Node-link and matrix graph visualization have been already contrasted to each other, but only for single types of edges: Ghoniem et al. [8], [9] evaluated these using seven simple, generic tasks as well as different sizes and different densities of random undirected graphs. Their results indicate that node-link diagrams are more suitable for most tasks for smaller graphs while matrix visualizations are more readable for large ones. An exception forms the task of finding paths where node-link diagrams perform better or at least comparable. Keller et al. [10] extended this work by performing a similar study, still with simple, domain-independent tasks but on non-random data: they largely confirmed the previous results but pointed out that relying on the model, and even on personal preference, either representation can be advantageous.

In contrast to these studies, we addressed a more specific visualization scenario in a more complex, realistic application also including the distinction of multiple types of edges. The results of our study extend the previous studies, in particular, with respect to performing higher-level tasks such as finding graph structures: although low-level tasks showed quite different characteristics for node-link and matrix, the high-level tasks we studied did not provide any considerable differences between the contrasted approaches.

VII. CONCLUSIONS

The conducted explorative user study evaluated two approaches for comparing different types of edges in graph diagrams. In a realistic scenario, eight participants analyzed code couplings of software systems both based on static images as well as on interactive visualizations in an 82 minutes (on average) within-subject lab experiment. The choice between the two visualization approaches in this application
mainly seems to be a matter of personal preference as the two approaches—though based on opposing paradigms—did not show any basic differences in our study: the participants were able to identify equivalent structures in the presented graphs, addressed the same software engineering tasks with the interactive tools, and rated the usefulness of the approaches alike. In particular, vertices with a high degree, groups of similar edges within packages and between different packages were detected by most participants with both approaches. With respect to the targeted area of application in software engineering, the visualizations seem to be most suitable for program comprehension, namely, for finding central classes.

VIII. ACKNOWLEDGEMENTS

We thank all the participants of our explorative study for their time and commitment. Further, we would like to thank the anonymous reviewers for their comments on this paper. Since research was partially supported by the IRTG 1131 that is funded by the DFG (German Science Foundation) and the BMBF (project ViERforES).

REFERENCES