ClassViz: From Inspection Tool to Research Vessel

Satrio Adi Rukmono[1]

[1]Eindhoven University of Technology (TU/e), De Zaale, Eindhoven, The Netherlands

Abstract

This work explores how visualisation bridges precise but complex software representations and the needs of human comprehension. We present *ClassViz*, a prototype that began as a lightweight inspection aid for code-to-graph instantiations and evolved through student projects, industrial collaborations, and research integration into a versatile platform for exploring software structures. Its trajectory illustrates how even a modest tool can become a shared environment for experimentation, evaluation, and communication.

Keywords

software architecture visualization, software maintenance, program comprehension

1. Introduction

Visualisation is central to understanding and maintaining software systems, complementing textual and structural representations. Graph-based models of source code are precise but often too large for direct inspection. In prior work, we proposed [1] labelled property graph (LPG)-based representations of software entities, relations, and attributes in a schema-light format [2]. Visualisation provides an intermediate form for inspecting instantiation outputs and communicating insights.

ClassViz[1] was developed to support such inspection, evolving into a platform used in research, education, and industry. It featured in the industrial evaluation of our deductive software architecture recovery (DSAR) technique [3, 4], acting as the "exoskeleton" for explanatory artefacts shown to participants.

2. Related Work

Software visualisation has addressed aspects of structure, behaviour, and evolution. A systematic review by Chotisarn et al. [5] of 105 articles (2013–2019) found that visualisations primarily target design, implementation, and maintenance tasks, typically using multivariate, graph-based, or metaphorical encodings (e.g., cities). Despite this diversity, industrial uptake remains limited, especially for maintenance and debugging. This underscores the need for practical and usable tools.

In software architecture visualisation, Shahin et al. [6] surveyed 53 studies (1999–2011). They identified four main types: graph-based, notation-based, matrix-based, and metaphor-based. Graph-based approaches dominate recovery and evolution tasks; metaphor-based views offer intuitive overviews, but raise scalability and cognitive concerns. Most techniques were tested only on small or academic systems, with little industrial use. The review emphasised the dual role of architecture visualisation: supporting both structural viewpoints (components, connectors, layers) and decisional viewpoints (design decisions and rationale).

These reviews frame our visualisation approach. ClassViz supports structural analysis of instantiated architecture graphs with filtering, layout, and layering. Its adoption in academic and industrial settings illustrates one response to calls for usable and practice-oriented tools.

BENEVOL'25: The 24th Belgium-Netherlands Software Evolution Workshop, November 17–18, 2025, Enschede, The Netherlands

s.a.rukmono@tue.nl (S. A. Rukmono)
https://satrio.rukmono.id/ (S. A. Rukmono)
0000-0001-9480-7216 (S. A. Rukmono)

[1]Live and loaded with JHotDraw 5.1 example input: https://satrio.rukmono.id/classviz/?p=jhotdraw-5.1

3. Visualisation Approach

ClassViz began as a pragmatic aid for visually inspecting code-to-graph instantiations [1]. It evolves through cycles of feedback, extension, and integration. Rather than a fixed set of requirements, its growth was driven by recurring needs arising from ongoing research. These needs form the basis for three research questions.

RQ1 What *visual affordances* support effective *lightweight inspection of labelled-property code graphs* for correctness and usability assessment?

RQ2 What *factors* influence the *adoption, extension, and appropriation of software structure visualisation tools* in educational and industrial contexts?

RQ3 How can software visualisation tools be designed to serve as *effective frontends* for diverse *automated analyses* such as architectural recovery and summarisation?

3.1. RQ1: Lightweight Visual Inspection of Graphs—Our Motivation and Origin

The initial version of ClassViz focused on inspecting large and complex graph instantiations produced by our tool Javapers[2]. These graphs were precise, but overwhelming, too large for immediate feedback. The initial design therefore emphasised clarity and filtering to produce diagrams that could be inspected during iterative development.

In ClassViz, nested boxes represented packages and classes, with class-level relations drawn as coloured UML-style line-with-end symbols (e.g., hollow triangles for inheritance, diamond for composition). Filtering allowed toggling of primitive types, packages, and relation types, and highlighting of classes by name. Click-through navigation revealed related elements, supporting localised exploration without overwhelming the view. Relations could be rendered as orthogonal or Bézier lines, with a choice of layouts from general-purpose algorithms. Figure 1 shows the initial version of ClassViz applied to JHotDraw 5.1.

These design choices enabled rapid inspection during the development of our LPG instantiation tool, allowing correctness and usability checks without heavy tooling. ClassViz offers minimal but expressive visual affordances—nested boxes, UML-style arrows, filters, and highlighting—that reduce cognitive load while preserving structural fidelity. Implemented as a lightweight browser application, it runs easily during short development cycles. **These simple encodings, filtering, navigation, and infrastructure made it an effective inspection aid, enabling quick structural sanity checks and supporting practical debugging of graph instantiation quality throughout development.**

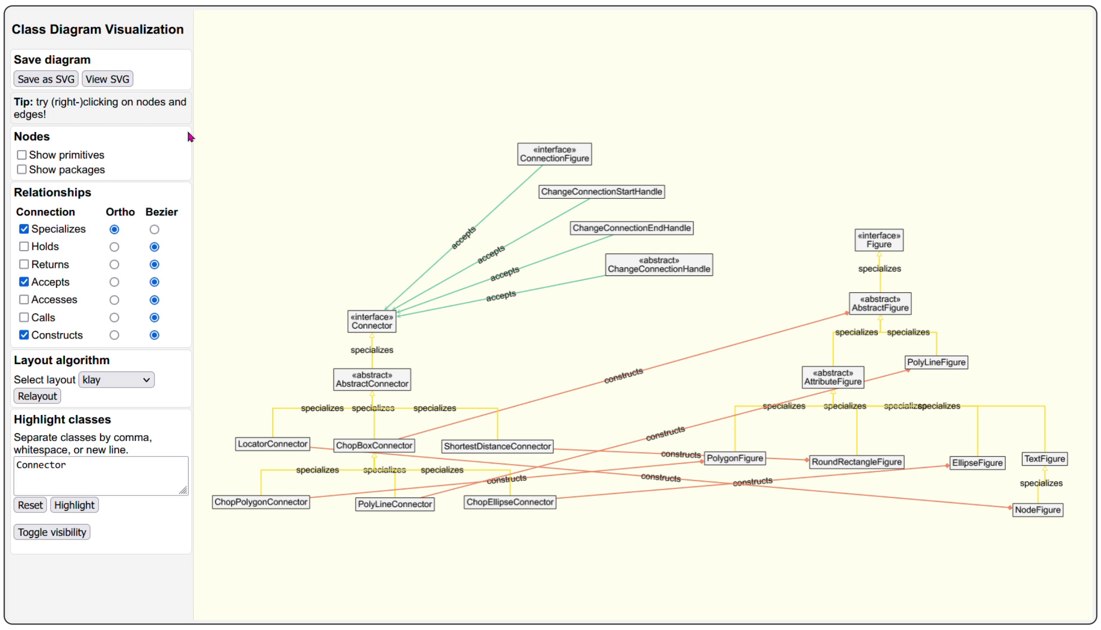
3.2. RQ2: Accessibility and Modifiability in Educational and Industrial Contexts

ClassViz's lightweight, browser-based architecture and minimal infrastructure assumptions made it easy to adopt and extend. Key factors that influenced its uptake in educational and industrial contexts include: (i) low barrier to entry for developers, (ii) openness to diverse input/output semantics in the same syntax, and (iii) modifiability by design. These qualities enabled it to function not just as a tool but as a shared platform across multiple independent student projects and industry-facing efforts.

Several student extensions explored dynamic behaviour visualisation. Fung [7] added overlays for execution traces; Tanis [8] built a summarisation frontend for large-scale traces; He [9] combined static structure with activity overlays; and Van Esch [10] brought ClassViz concepts to virtual reality, combining structure and dynamic sequence data in 3D. Together, these works show how runtime-oriented adaptations were feasible across fidelity levels and modalities.

Other works focused on improving structural clarity and visual encoding. Kloet [11] addressed the legibility and usefulness of applying distinct layouts across abstraction levels. *BubbleTea* and *CodeView* [12, 13] introduced alternative layout metaphors: layered bubble-packing and abstracted

[2]https://github.com/rsatrioadi/javapers


Class Diagram Visualization

Save diagram

Save as SVG | View SVG

Tip: try (right-)clicking on nodes and edges!

Nodes

☐ Show primitives
☐ Show packages

Relationships

Connection	Ortho	Bezier
☑ Specializes	●	○
☐ Holds	○	●
☐ Returns	○	●
☑ Accepts	○	●
☐ Accesses	○	●
☐ Calls	○	●
☑ Constructs	○	●

Layout algorithm

Select layout: klay

Relayout

Highlight classes

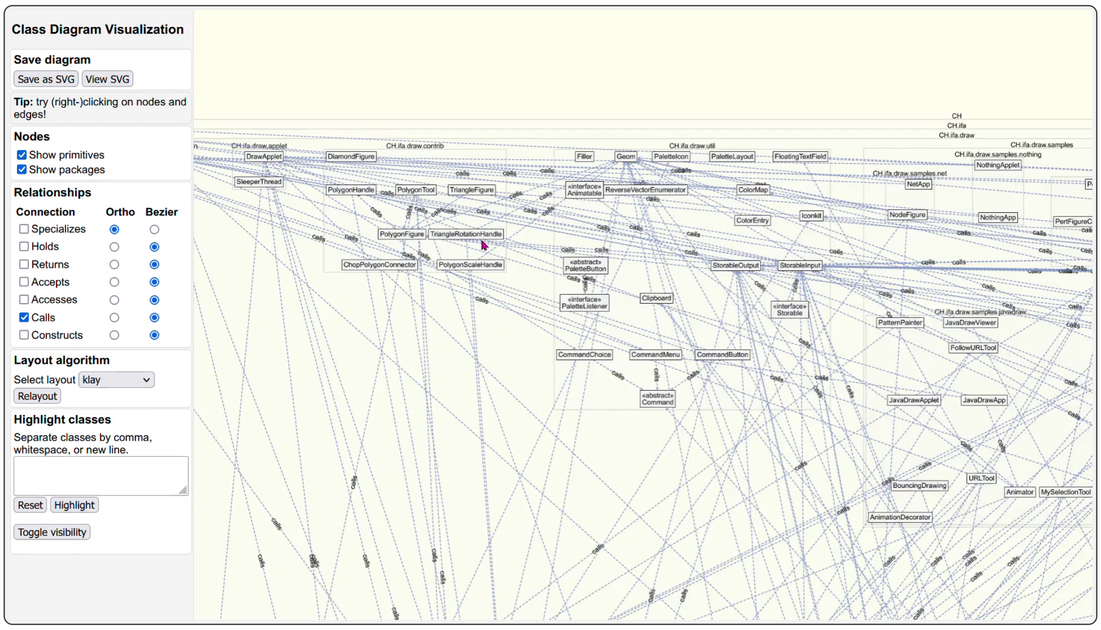
Separate classes by comma, whitespace, or new line.

```
Connector
```

Reset | Highlight

Toggle visibility

(a) Classes involved in an instance of the strategy design pattern in JHotDraw 5.1. Other classes are filtered out.


Class Diagram Visualization

Save diagram

Save as SVG | View SVG

Tip: try (right-)clicking on nodes and edges!

Nodes

☑ Show primitives
☑ Show packages

Relationships

Connection	Ortho	Bezier
☐ Specializes	●	○
☐ Holds	○	●
☐ Returns	○	●
☐ Accepts	○	●
☐ Accesses	○	●
☑ Calls	○	●
☐ Constructs	○	●

Layout algorithm

Select layout: klay

Relayout

Highlight classes

Separate classes by comma, whitespace, or new line.

Reset | Highlight

Toggle visibility

(b) All packages and classes of JHotDraw 5.1 (that fit the viewport) and the “calls” relations among the classes.

Figure 1: Screenshots from an early ClassViz version, showing JHotDraw 5.1 in different modes.

views, respectively, while Atisomya [14] explored direct mapping of analytical dimensions to visual variables such as position and colour, showcasing the flexibility of the underlying design space.

ClassViz also served as a frontend for specialised analysis pipelines. Asuni [15] added overlays for vulnerability detection results. Kakkenberg et al. [16] applied ClassViz principles to low-code platforms, resulting in *Arvisan*, an industrially evaluated tool that continues to be useful in different domains [17].

Notably, the simplicity of ClassViz brings operational limitations that encouraged parallel evolution. Morier[3] developed a shared backend to support authentication, graph versioning, and permission-aware data access, laying the groundwork for a unified ecosystem across ClassViz forks and related tools.

Overall, these projects demonstrate that ClassViz’s adoption and extension were driven not by feature completeness but by a deliberately minimal and open design that enabled diverse appropriation.

[3]https://github.com/SimonMorier/ArchManager-back

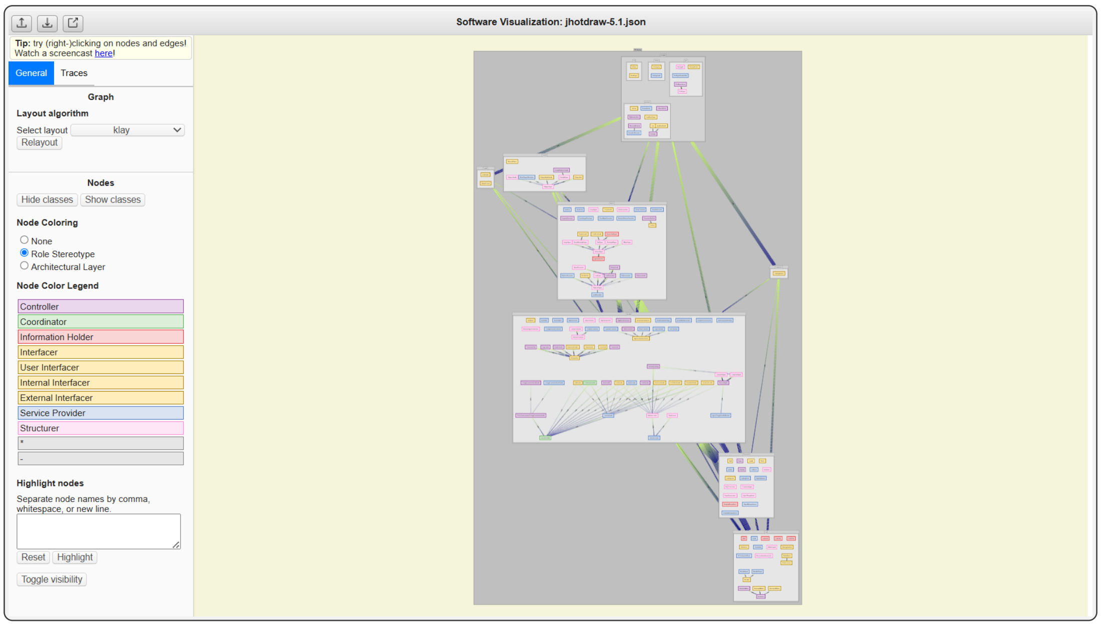
The low entry barrier and minimal coupling empowered non-core developers to repurpose ClassViz across adjacent (sub)domains without deep reengineering.

3.3. RQ3: Visual Frontend for Architectural Analysis and Explanation

Feedback from students and collaborators led to iterative adaptations that broadened ClassViz's functionality and transformed it into a shared research artefact. A central enabler of this evolution is its use of LPGs, which allow analysis results to be integrated flexibly, either as additional nodes/edges or as properties on existing ones. These properties can be directly mapped to visual variables, e.g., metrics to colour gradients, classifications to discrete colours, and ranked values to spatial layout (e.g., vertical position by layer). This makes ClassViz well-suited as a frontend for structurally anchored explanations.

ClassViz was used as the presentation layer for hierarchical summarisation [18] and DSAR [3] outputs. To support these use cases, we added node colouring for classifications (i.e., role stereotypes [19], architectural layers [20]) and a detail pane for automatically generated summaries. These enhancements grounded abstract analysis results in the system's structure, improving interpretability and traceability.

Additional enhancements, some drawn from extension projects, were integrated to further support explanations. These include lifting and lowering relations across abstraction levels, gradient and thickness styling for edges to encode direction and weight, and more intuitive filtering and navigation. Figure 2 shows a recent version of ClassViz with role-classification results visualised using colour-coded nodes.


Software Visualization: jhotdraw-5.1.json

Tip: try (right-)clicking on nodes and edges! Watch a screencast here!

Select layout: klay

Node Coloring: ( ) None [✓ Role Stereotype] ( ) Architectural Layer

Separate node names by comma, whitespace, or new line.

Figure 2: A recent version of ClassViz showing classes from JHotDraw 5.1 classified into role stereotypes [19] and call relation lifted into package level.

ClassViz was then deployed in an industrial evaluation at ASML. Our study [4], under review at ICSE-SEIP, assessed the tool in context: ClassViz acted as the explanatory surface for DSAR-derived architectural views. What began as a simple tool for visual inspection evolved into a presentation layer for explanation workflows, supporting interpretation and communication in both research and industry settings. **These integrations show how LPGs bridge abstract analysis results and architectural explanations by enabling node and edge properties to be directly mapped to visual variables.**

3.4. Eating Our Own Dog Food: Visualising ClassViz in ClassViz

Figure 3 shows the internal structure of the ClassViz source code, rendered in ClassViz itself with a manually arranged layout and minor post-processing for legibility. The diagram combines two abstraction levels: filesystem folders, shown in UML-style package notation; and JavaScript modules, shown as labelled rectangles. Modules are coloured by DSAR-inferred role stereotype (e.g., *Controller*, *Coordinator*); folders by architectural layer (i.e., *Presentation*, *Domain*, *Data*). Although ClassViz normally toggles between these modes, the figure overlays both to avoid duplication.

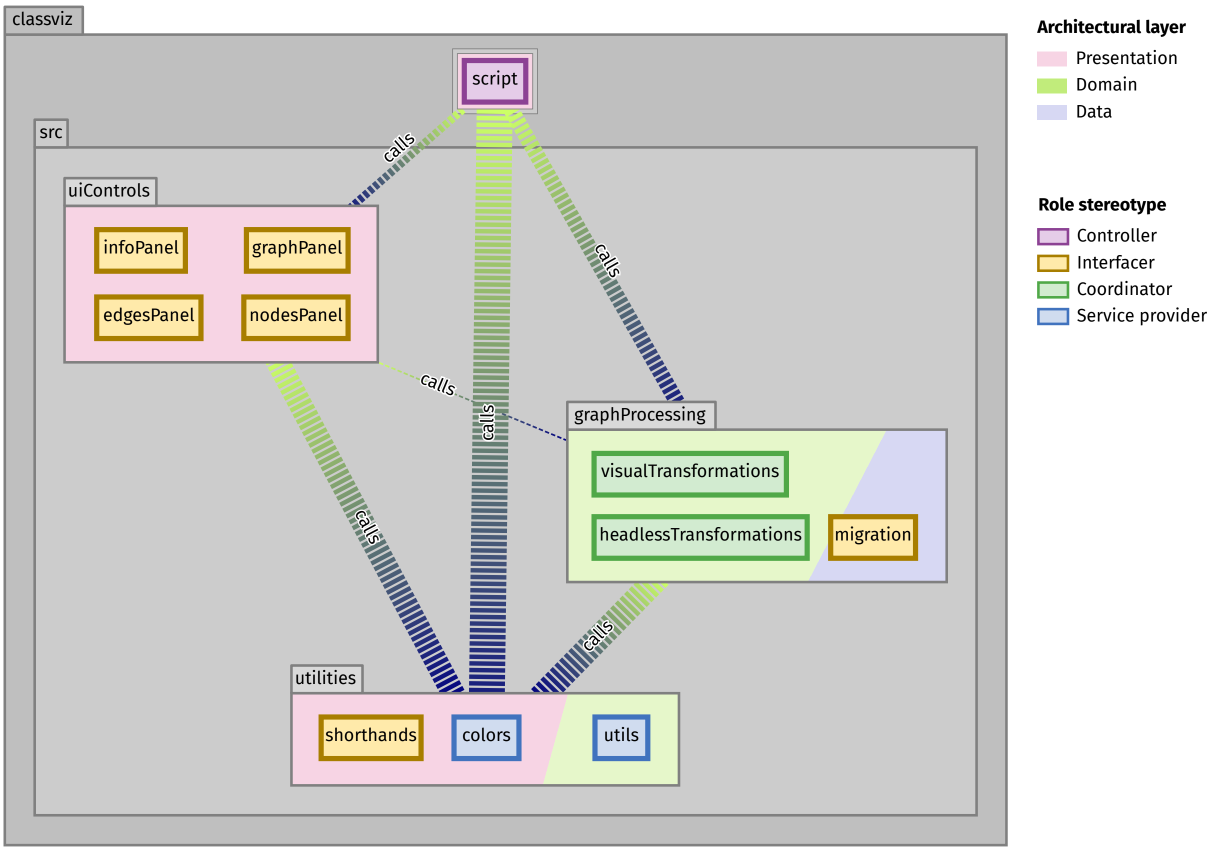


Figure 3: ClassViz source code organisation as depicted by ClassViz itself.

The graphProcessing folder implements the core pipeline, converting graphs into visual form by collapsing classes, assigning visual cues, and filtering edges. The uiControls folder handles user interaction, and utilities provides shared functions such as colour assignment.

This self-visualisation validates the tool on a non-trivial codebase and also offers design feedback: the dominance of calls to utility functions suggests overcentralisation, while the segmentation of UI panels highlights interface modularity.

4. Reflection

The trajectory of ClassViz shows how a pragmatic artefact can evolve into a central research instrument. Its growth was shaped by shifting research needs, with each inquiry prompting adaptations and, in turn, new questions. Visualisation tools in software engineering are rarely static; ClassViz illustrates how adaptability enables experimentation, pattern discovery, and explanation.

This adaptability stems from deliberate design minimalism and architectural openness. By avoiding rigid assumptions, ClassViz remained easy to extend, supporting overlays for dynamic behaviour, security, stereotypes, and VR with minimal friction. Its forks show that modifiability often outweighs completeness. More broadly, explanation and inspection are not auxiliary but *generative*; seeing structure often precedes explaining it, making explanatory tooling a valid research contribution.

References

[1] S. A. Rukmono, M. R. Chaudron, Enabling Analysis and Reasoning on Software Systems through Knowledge Graph Representation, in: 20th International Conference on Mining Software Repositories, IEEE, 2023, pp. 120–124. doi:10.1109/MSR59073.2023.00029.

[2] D. Anikin, O. Borisenko, Y. Nedumov, Labeled Property Graphs: SQL or NoSQL?, in: 2019 Ivannikov Memorial Workshop (IVMEM), IEEE, 2019, pp. 7–13.

[3] S. A. Rukmono, L. Ochoa, M. R. Chaudron, Deductive Software Architecture Recovery via Chain-of-Thought Prompting, in: 44th International Conference on Software Engineering: New Ideas and Emerging Results, Association for Computing Machinery, New York, NY, USA, 2024, pp. 92–96. doi:10.1145/3639476.3639776.

[4] S. A. Rukmono, L. Ochoa, T. M. Bressers, J. Krüger, M. R. Chaudron, Evaluating Explanatory Artefacts of DSAR-Recovered Architectures from Industrial Codebases, 2025.

[5] N. Chotisarn, L. Merino, X. Zheng, S. Lonapalawong, T. Zhang, M. Xu, W. Chen, A Systematic Literature Review of Modern Software Visualization, Journal of Visualization 23 (2020) 539–558.

[6] M. Shahin, P. Liang, M. A. Babar, A Systematic Review of Software Architecture Visualization Techniques, Journal of Systems and Software 94 (2014) 161–185.

[7] K. Y. Fung, Classifying Java Classes Into Role Stereotypes Based on Their Behavior, Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2023.

[8] H. Tanis, LLM-Enhanced Code Comprehension by Combining Static and Dynamic Analysis in Large-Scale C++ Systems, Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2025.

[9] Y. He, Enhancing Program Comprehension through Visualization and LLM-based Summarization of Behaviour, Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2025.

[10] H. v. Esch, Visualizing Software Behavior & Structure in Virtual Reality, Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2024.

[11] P. Kloet, Multi-level Layout Algorithms for Visualizing Hierarchical Software Systems, Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2024.

[12] S. A. Rukmono, M. R. Chaudron, C. Jeffrey, Layered BubbleTea Software Architecture Visualisation, in: Working Conference on Software Visualization, IEEE, 2024, pp. 122–126. doi:10.1109/VISSOFT64034.2024.00024.

[13] C. Jeffrey, A. P. Nugroho, S. A. Rukmono, Y. Widyani, CodeView: A Tool for Software Visualization in Development View, in: 2024 IEEE International Conference on Data and Software Engineering (ICoDSE), IEEE, 2024, pp. 67–72.

[14] A. K. Atisomya, Pengembangan Alat Visualisasi Arsitektur Perangkat Lunak untuk Analisis Multidimensi, Bachelor's thesis, Institut Teknologi Bandung, Sekolah Teknik Elektro dan Informatika, Bandung, Indonesia, 2025.

[15] M. Asuni, Effective Graphical Visualization of Vulnerabilities in C and C++ Programs, Master's thesis, University of Cagliari, Faculty of Engineering and Architecture, Cagliari, Italy, 2024.

[16] R. Kakkenberg, S. A. Rukmono, M. R. Chaudron, W. Gerholt, M. Pinto, C. R. de Oliveira, Arvisan: an Interactive Tool for Visualisation and Analysis of Low-Code Architecture Landscapes, in: Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, 2024, pp. 848–855.

[17] F. Zamfirov, A. Radulescu, J. Krüger, M. R. Chaudron, Lessons from Visualizing Software Architecture Structure Conformance at Thermo Fisher Scientific, in: seaa, springer, 2025. doi:10.1007/978-3-032-04207-1_25.

[18] S. A. Rukmono, L. Ochoa, M. R. Chaudron, Achieving High-Level Software Component Summa-

rization via Hierarchical Chain-of-Thought Prompting and Static Code Analysis, in: International Conference on Data and Software Engineering, IEEE, 2023, pp. 7–12. doi:10.1109/ICoDSE59534.2023.10292037.

[19] R. Wirfs-Brock, A. McKean, I. Jacobson, J. Vlissides, Object Design: Roles, Responsibilities, and Collaborations, Pearson Education, 2002.

[20] M. Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2012.