A Comparison of Visualization Concepts and Tools for Variant-Rich System Engineering

Siyue Chen
Eindhoven University of Technology
Eindhoven, The Netherlands
s.chen1@student.tue.nl

Loek Cleophas
Eindhoven University of Technology
Eindhoven, The Netherlands
Stellenbosch University
Stellenbosch, South Africa
l.g.w.a.cleophas@tue.nl

Jacob Krüger
Eindhoven University of Technology
Eindhoven, The Netherlands
j.kruger@tue.nl

ABSTRACT

Software product-line engineering is concerned with developing a set of similar, yet customized, software systems that share a common codebase. To develop such a variant-rich system, various development processes, techniques, and tools have been studied in research and are used in practice. Specifically, to help developers manage the complexity of developing large-scale variant-rich systems, researchers have proposed visualizations to visually present different properties of such systems and their engineering—such as feature models, configurations, the similarity of variants, or process traces. Two recent mapping studies have systematically elicited the state-of-the-art on such visualizations, but neither of them provides a comparative analysis of the underlying visualization concepts and tools. In this paper, we report a qualitative meta-analysis of the 64 papers that we primarily selected from these two mapping studies. Advancing on the previous studies, we compare the use cases, pros, cons, and relations between visualization concepts and tools used with respect to engineering variant-rich systems. Our results provide insights—orthogonal to those from the mapping studies—regarding the purposes for which visualization concepts are used and the tools that are available to implement these concepts. The overview we provide can help researchers as well as practitioners decide to use specific established visualization concepts or design new ones, and identify tools that can help them to implement these.

CCS CONCEPTS

• **Software and its engineering** → **Software product lines**; • **Human-centered computing** → **Visualization techniques**; **Visualization systems and tools**.

KEYWORDS

software product line, variant-rich system, visualization techniques, visualization systems and tools

ACM Reference Format:
Siyue Chen, Loek Cleophas, and Jacob Krüger. 2023. A Comparison of Visualization Concepts and Tools for Variant-Rich System Engineering. In *27th ACM International Systems and Software Product Line Conference - Volume A (SPLC '23), August 28-September 1, 2023, Tokyo, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3579027.3608986

SPLC '23, August 28-September 1, 2023, Tokyo, Japan

ACM ISBN 979-8-4007-0091-0/23/08.
https://doi.org/10.1145/3579027.3608986

1 INTRODUCTION

Software product-line engineering provides means for engineering a set of similar systems that build on a common codebase [4, 53, 82], thereby establishing a *variant-rich system* (a.k.a. product family or product line). Since most of today's software systems have to exist in various variants to account for individual customer requirements, variant-rich systems are widely established in different domains, such as automotive, aerospace, and embedded firmware [29, 51, 52, 55–58, 79, 91, 100, 103]. Typically, developers initiate a variant-rich system via the clone-and-own method of copying and adapting a system to new requirements (e.g., via forking on social-coding platforms) and move to a fully integrated platform later on [21, 28, 49, 55, 86, 91]. An integrated platform requires more planning and investment in advance [1, 13, 17, 50, 51, 89], for instance, to set up the platform architecture, feature model [44, 75], variability mechanism [14], pipelines for configuring and deriving variants [18, 97], as well as testing [24]. However, the benefits (e.g., quality improvements, faster time-to-market) of a platform implemented based on product-line engineering methods usually outweigh the initial investments, which is why most organizations that develop a variant-rich system adopt a platform at some point [13, 51, 89, 100].

Regardless of how they were developed or are organized, real-world variant-rich systems (e.g., the Linux Kernel [94]) can become very large, for instance, in terms of their features, variants, source code, or the number of developers involved. Visualizations are key to helping developers understand and manage the size, complexity, as well as relations of such entities (e.g., feature dependencies, evolution graphs, fork networks). A visualization can provide a clear and intuitive representation of a certain part of a variant-rich system and its properties. Two independent mapping studies [61, 68] have been conducted recently to provide overviews of existing visualizations used in product-line engineering. Both focus on the entities that are visualized and the types of diagrams used. However, neither one reports a comparative analysis of the use cases for which the visualization concepts and respective tools are used. For instance, Medeiros et al. [68] provide an overview of what diagrams are used, what evolution scenarios [93] are covered, what benefits are reported, or what users can interact how with a visualization. In contrast, we are interested in what visualization concepts (e.g., diagrams) have been used for what reasons to visualize a certain entity of engineering variant-rich systems (e.g., features, forks) and the respective tools used. More specifically, we are not interested in the evolution scenarios, but, for example, why different visualizations like trees, graphs, or logical gates are used to represent feature dependencies. So, we aim to complement the two mapping studies with orthogonal insights that are important for researchers and practitioners to decide on using or designing visualizations for engineering variant-rich systems.

In this paper, we present a conceptual comparison of the different visualization concepts proposed and used in the context of engineering variant-rich systems. For this purpose, we performed a meta-analysis of 64 papers from which we elicited the visualization concepts, the visualized entities, the researchers' underlying ideas, the evaluations performed, and the tools used. We discuss pros and cons of the underlying visualization concepts, relations, and tools, providing examples of their application in research and practice.

More precisely, we contribute the following in this paper:

- We provide an overview of what visualization concepts have been used for what entities in the context of engineering variant-rich systems.
- We analyze what use cases researchers have aimed to support using specific visualization concepts, as well as the tools they used for this purpose.
- We discuss what visualization concepts have been used, and what evidence for their usefulness has been reported.
- We share our analysis dataset with more detailed descriptions in a persistent open-access repository.[1]

Building on our contributions, researchers as well as practitioners can more easily identify appropriate concepts and tools to support their own use cases or to develop novel visualization ideas.

2 RELATED WORK

There are two recent mapping studies on the various visualization concepts that have been proposed for engineering variant-rich systems, which are the work most closely related to our own analysis.

Lopez-Herrejon et al. [61] have conducted a systematic mapping study on visualizations for software product-line engineering, summarizing findings for 37 primary sources. They performed their meta-analysis by gathering information on the visualization techniques and providing a categorized overview of the findings. Specifically, Lopez-Herrejon et al. have identified ten (types of) visualization tools (cf. Table 3), which they briefly introduce. However, the review does not explain and analyze what and why visualization concepts and tools are used for specific visualizations. The results also do not explain whether specific concepts or techniques are more feasible for certain use cases or entities of engineering variant-rich systems.

Medeiros et al. [68] report a systematic mapping study of 41 primary sources that propose visualizations for evolving variant-rich systems. The authors conducted their study to tackle three primary research questions that are structured around established evolution scenarios [93]: what analyses are conducted, what visualizations are displayed, and how mature are the techniques? Medeiros et al. provide in-depth insights into individual visualizations, elicit what entities of variant-rich systems engineering are visualized (e.g., features, variants), what concepts (e.g., bar chart, tree-map, table) are used, and that only 13 tools were publicly available. Unfortunately, the analysis investigates only how often visualization concepts and entities of engineering variant-rich systems have been considered, but does not investigate the relations between both. Moreover, an overview of the underlying tools available for researchers and practitioners is missing.

3 METHODOLOGY

In this section, we describe the methodology we employed for our comparative meta-analysis.

3.1 Goal and Research Questions

With this study, we aimed to extend the findings of the two mapping studies we described in Section 2 with orthogonal insights. In particular, we aimed to understand what visualization concepts have been used for what entities of engineering variant-rich systems, and what tools have been used for what reasons. For instance, neither of the mapping studies has analyzed why feature models have not only been visualized as typical trees, but also using concepts like tables, bubble maps, or scatter charts. We are concerned with exploring such relations between visualization concepts and entities, and with understanding what evidence exists that the visualizations are helpful for developers.

To tackle this goal, we defined three research questions (RQs):

RQ_1 *What visualization concepts have been used for what entities?* Some visualization concepts are widely established for specific entities, for instances trees (concept) to display feature models (entity) as diagrams. However, variant-rich systems exhibit various entities with relationships, which is why various visualization concepts have been proposed and combined. We elicited such combinations to provide an overview of the paths that have been explored.

RQ_2 *What are the ideas of using these visualizations?* Each visualization builds on an underlying idea for its specific use case. For instance, trees for feature models are the de facto standard (idea) [75], but some researchers proposed adding cone trees to provide a better visualization of how multiple variant-rich systems are connected. Exploring the underlying ideas of each visualization contributes to an understanding of what entities and relations the respective researchers aimed to visualize.

RQ_3 *What is the evidence on the usefulness of these visualizations?* When building on existing or proposing new visualizations, it is key to understand whether they actually help developers. For instance, while 3D cones have been used for visualizing feature models, their actual usefulness for developers has not been empirically evaluated (no insights). We collected an overview of which visualization concepts and underlying ideas have proven to benefit developers.

RQ_4 *What tools have been used for implementing visualizations?* To develop a visualization, it is helpful to build on established visualization tools. As an example, FeatureIDE [69] is a widely established tool for visualizing feature models, but extending it with new visualizations is quite cumbersome. To support researchers and practitioners in designing new visualizations, we provide an overview of the tools used in previous work that they can build upon.

[1] https://doi.org/10.5281/zenodo.8069277

These research questions were outside the scope of the two mapping studies we build upon. Consequently, we argue that we contribute orthogonal and valuable in-depth insights into visualizations for engineering variant-rich systems.

3.2 Selecting Primary Sources

For our meta-study, we relied on the papers identified in the two systematic mapping studies by Lopez-Herrejon et al. [61] and Medeiros et al. [68]. Both works conducted systematic searches for relevant papers: Lopez-Herrejon et al. conducted an automated search on ScienceDirect, IEEEXplore, the ACM Digital Library, and SpringerLink, followed by a snowballing search. This resulted in 37 papers published between 2007 and 2016. Medeiros et al. performed an automated search on ScienceDirect, IEEEXplore, the ACM Digital Library, Scopus, and the Wiley Online Library, also followed by a snowballing search. They identified 41 papers that have been published from 2006 until 2021. Overall, this constituted a total of 58 papers after removing the duplicates between the two studies.

As a cross check, particularly for more recent papers, we performed a manual literature search through the 2016 to 2022 proceedings of the International Conference on Visualisation (VIS), International Working Conference on Software Visualisation (VISSOFT), and the International Systems and Software Product Lines Conference (SPLC). This process led to six new papers [8, 37, 38, 63, 72, 73] that we added to the previous ones, adding up to a total of 64 papers that we considered for our analysis. During our manual search, we selected any paper as relevant if it (i) has been published at the research track and (ii) reports on a visualization for engineering variant-rich systems. For this purpose, we first checked whether we could decide based on the title, then we checked the abstract, and if needed we read the full text of a paper. Since we investigated three prominent conferences, all papers fulfilled the typical criteria of being peer-reviewed and written in English.

3.3 Data Extraction

To answer our research questions, we extracted the following data (besides typical bibliographic data):

- **RQ_1** A list of keywords summarizing the entities that are visualized (e.g., features, feature model, variants).
- **RQ_1** A list of keywords describing the concepts used for visualizing (e.g., bar chart, tree, histograms).
- **RQ_1** Short descriptions of each relation between entities and concepts (i.e., short notes connecting one or more entities with one or more concepts).
- **RQ_2** The idea or use case for which each relation has been proposed for a visualization (i.e., a short note for each relation, such as "*de facto standard of using trees for feature models*").
- **RQ_3** Information on the evaluations that have been performed (e.g., experiment with 10 students solving tasks).
- **RQ_3** Any results of the evaluation reported in the paper (e.g., feedback, task-solving performance).
- **RQ_4** A list of all tools used for implementing a visualization together with short notes on the pros and cons of these tools reported in a paper.

Using this data, we were able to answer our research questions and provide novel insights compared to the related work.

Table 1: Visualization concepts used in the primary sources.

Concept	Sources	Total
Bar chart	[20–22, 54, 66, 71]	6
Bubble chart	[60, 74]	2
Colored code	[15, 45, 71]	3
Concept lattice	[39, 59, 78]	3
Cone tree	[98]	1
Feature blueprint	[99]	1
Graph	[2, 3, 5, 15, 16, 25, 32, 34, 35, 40, 62, 67, 72, 73, 80, 81, 84, 87, 95, 104]	20
Heatmap	[8, 26, 60, 78]	4
Histogram	[92]	1
Levelized structure map	[46]	1
Line chart	[25]	1
Logic gate	[31]	1
Sankey diagram	[70, 71, 81]	3
Scatter chart	[12]	1
Survival chart	[101]	1
Table/Matrix	[2, 7, 15, 25, 30, 36, 47, 60, 74, 83, 84, 88]	12
Tree	[3, 6, 8, 10, 11, 27, 31, 37, 38, 41–43, 63, 76, 90, 92, 96]	17
Treemap	[40, 60, 70, 72, 73]	5
Word cloud	[19, 65]	2

3.4 Data Analysis

To ensure consistency, the first author extracted all data by reading each paper in detail. They employed an open-coding process, marking relevant information in the full texts and extracting them into a spreadsheet. The other authors of this paper verified the extracted data during discussions, data analyses, based on their knowledge of these papers, and by picking individual papers for detailed checks. We then analyzed our data qualitatively, using open-card-sorting to derive common themes and terminologies from it.

4 RESULTS

We identified 19 visualization concepts from the primary sources. In Table 1, we display the distribution of what visualization concept has been used in the primary sources; and in Figure 1, we display the entities of engineering a variant-rich system that have been visualized using these concepts (**RQ_1**). To understand the selection of visualization concepts used in research, we explored the reasoning behind these concepts (**RQ_2**). Next, we provide a brief overview of the concepts, their underlying ideas, the existing evidence (**RQ_3**), and the tools used (**RQ_4**)—with more detailed descriptions in our dataset,[1] answering our research questions in combination to avoid redundancy and improve the comprehensibility of this section.

*Answering **RQ_1**: Visualization Concepts Used.* We identified 19 visualization concepts that have been used for eight different entities. While trees (17) and graphs (20) are the most commonly used visualization concepts, due to the hierarchical organization of variant-rich systems, there are a variety of other concepts that have been proposed to more effectively communicate data and insights. For hierarchical data like feature models and software artifacts, trees and treemaps are commonly used. Chronological data, such as changes in the lines of code of a feature over time, are typically visualized using tables, matrices, or survival charts. Categorized data, such as features, have been visualized using various concepts. In contrast,

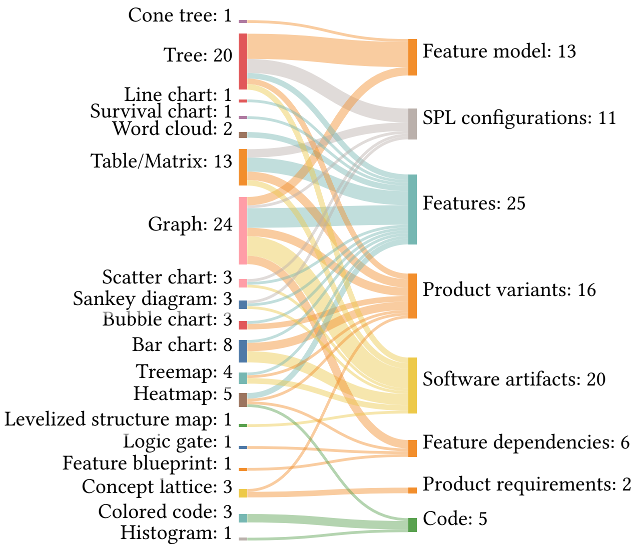

Figure 1: Visualization concepts (left) and the entities they visualize (right). The numbers indicate how often we identified each concept or entity. Note that these numbers do not precisely match those in Table 1, because some concepts/entities are used multiple times in one paper, for different purposes.

word clouds and heatmaps are rather specific and can indicate the importance or relevance of a particular topic or question.

*Answering **RQ$_2$**: Ideas of Visualizations.* Some visualization concepts used for engineering variant-rich systems, such as trees, graphs, tables, or matrices, are generally used for visualizing feature models and their derivatives (features and feature dependencies). In contrast, other visualization concepts have an underlying idea specific to their use case, such as colored code, word clouds, and concept lattices. The remaining visualization concepts have been explored for visualizing various entities of a variant-rich system, such as the overall structure, code complexity, or importance of features—typically associated with each visualization concept's ability to provide a unique perspective on the data. By using or combining different visualization concepts, entities within a variant-rich system can be associated with each other, which provides an understanding of the relationships or patterns in the system.

*Answering **RQ$_3$**: Evidence on Visualizations.* Various evaluation methods have been used in the primary sources, showing evidence on the benefits of the visualization concepts covered. In Table 2, we display the distribution of the four evaluation methods we identified from the primary sources. While field experiments are a common

Table 2: Evaluation methods used in the primary sources.

Evaluation	Sources	Total
Field experiment	[5–8, 10, 11, 20, 22, 26, 31, 32, 36, 38–40, 42, 43, 47, 54, 59, 63, 65–67, 72, 73, 76, 78, 80, 81, 83, 88, 98, 99]	34
Judgement study	[2, 71, 74]	3
Experimental simulation	[41, 95]	2
Field study	[101]	1

Table 3: Visualization tools used in our selected primary sources. The tools that have previously been identified by Lopez-Herrejon et al. [61] are asterisked (*).

	Tool	Sources	Total
*	Adhoc	[7, 8, 12, 15, 20–22, 26, 27, 31, 39, 46, 62, 64, 87, 90, 92, 98, 101, 104]	20
*	Eclipse EMF-GEF	[3, 10, 11, 25, 30, 32, 34, 35, 45, 66, 76, 80, 83, 84, 88]	15
*	Prefuse	[6, 41, 81]	3
*	D3.js	[19, 40, 47, 60, 95]	5
*	Graphviz	[42]	1
*	CCVisu	[5]	1
*	Google Charts	[74]	1
*	ConExp	[59]	1
*	Moose	[99]	1
*	Processing	[67]	1
	FeatureIDE	[2, 70]	2
	Babylon.js	[72, 73]	2

method used in visualization research, other methods, such as judgment studies, experimental simulations, and field studies, have also been employed to evaluate visualizations for variant-rich systems. From such evaluations, we found that a substantial body of evidence supports the notion that trees are an effective means for visualizing feature models, while graphs have been shown to be effective in visualizing features and software artifacts. Additionally, tables and matrices have been found to be effective in visualizing features. Other visualization concepts have, to some extent, been demonstrated to be effective in visualizing some use cases. However, the level of evidence supporting their effectiveness varies, and further research is needed to fully investigate their potential in different contexts. Finally, the 24 primary sources that we do not reference do not include any type of evaluation of their proposed visualizations, indicating a gap in the research and a need to further investigate the effectiveness of these visualizations.

*Answering **RQ$_4$**: Visualization Tools Used.* We found that the primary sources have used 12 different visualization tools: Eclipse GEF-EMF [85], Prefuse [33], D3.js [105], Graphviz [23], CCVisu [9], Google Charts,[2] ConExp [102], Moose [77], Processing[3], FeatureIDE [48, 69], Babylon.js,[4] and Adhoc tools) to implement their visualization concepts, even in the case of implementing the same concepts. In Table 3, we show what visualization tool has been used by which primary sources. Primary sources we do not list have introduced some visualization, but did not implement it within any tools. To figure out whether the choice of visualization tools is related to their supported capabilities, we analyzed the visualization concepts supported by each tool (details can be found in our dataset[1]). We define three levels of compatibility as follows:

Implemented. The visualization concept has been implemented using the visualization tool as reported in a primary source.

Supported. The visualization concept has not been implemented using the visualization tool as reported in the primary sources, but the concept is officially supported by the tool.

[2] https://developers.google.com/chart
[3] https://processing.org
[4] https://babylonjs.com

Not supported. The visualization tool is not designed to implement the visualization concept.

We gathered this overview by inspecting and testing each tool, as well as reading its documentation. To understand why researchers opted for a particular visualization tool over others, we analyzed the advantages and limitations of these tools in the context of engineering variant-rich systems, which we detail in the following. Note that "visualized entities" refers to the entities our primary sources have visualized with a tool, not the tools' general capabilities.

We found that some of the visualization tools are only designed for specific purposes: Graphviz and CCVisu are used to abstract graphs from structural information, while ConExp is specialized for formal concept analysis. Meanwhile, Prefuse and D3.js provide support for most visualization concepts, but have not been adopted to their full extent in the primary sources. We also note that although most of the primary sources do not explicitly point out their motivation for choosing a visualization tool, researchers tend to select the tool developed or supported by their institution (e.g., CCVisu [5] and Moose [99]). Thus, one factor that affects the choice of visualization tools could be familiarity with tools and languages. In our dataset,[1] we further provide an overview of the key specifications of the tools to help researchers and practitioners select the most suitable one for their context.

5 DISCUSSION

Visualizations play a crucial role in understanding variant-rich systems and communicating their complexities effectively. For **RQ$_1$**, we explored what and how visualization concepts have been used for different entities and found that, while some visualization concepts (e.g., trees, graphs) are commonly used, there are also entities for which other visualization concepts have been explored. We conclude that the choice of visualization concepts should be tailored to the specific data and perspectives being communicated.

For **RQ$_2$**, we analyzed the reasons for using a visualization concept for an entity. Some visualization concepts are generally used for visualizing specific entities in variant-rich systems, such as trees and graphs for feature models and their derivatives due to the hierarchical structure. Other concepts have an underlying idea specific to their use case, such as colored code, word clouds, and concept lattices. Our findings highlight the importance of selecting a suitable visualization concept or combination of techniques to communicate effectively, as well as of reflecting on standard visualizations and potential improvements.

For **RQ$_3$**, we examined the evaluation of each primary source. We found that there is reliable empirical evidence supporting the effectiveness of several visualization techniques used for variant-rich systems, such as trees, graphs, and tables/matrices for feature models and their derivatives. However, the level of evidence supporting the effectiveness of other visualization concepts varies, and most primary sources do not include an evaluation study. This clearly indicates the need for further research and systematic evaluations.

For **RQ$_4$**, we explored what tools the visualizations were implemented with. Some tools like Eclipse-based platforms are compatible with the development environment, enabling more advanced operations, such as code interactions and configuring. Researchers should carefully consider the available visualization tools and their specifications when selecting the most suitable tool for their requirements and ideas. We contribute an overview of the key specifications of each visualization tool, which can help researchers in selecting the most suitable tool for their requirements.

Our findings further reveal the heterogeneity of visualization tools regarding different use cases. Tools, according to their capabilities and limitations, handle tasks with different visualization techniques and integrate with workflows at different levels. In general, the selection of visualization tools is determined by the task domain, data type, and skills or experience of the user. Considering that skills and experience vary among users and we cannot take that into account, we assume that familiarity with the tools and programming languages does not impact a choice. Under that assumption, the following high-level considerations can help select proper tools for a visualization:

- *If the visualization must be embedded into a development workflow*, such as interacting with the source code or integrating click-and-configure experience, Eclipse EMF-GEF, FeatureIDE, Prefuse, and Adhoc tools seem most useful. Eclipse EMF-GEF and FeatureIDE work well when the variant-rich system is modeled and built with Eclipse; Prefuse can be integrated with Java projects; and Adhoc tools may be a better option to fit specific development environments.
- *If a formal concept analysis is needed*, ConExp is a proper choice specifically designed for this use case.
- *If software analyses on a complex variant-rich system are needed*, Moose can be an option for its capability to handle large-scale systems.
- *If feature-model visualizations shall be explored with more flexibility*, such as combining several visualization concepts in a view or applying tweaks on an existing visualization, the latest open-source visualization tools with proper documentation and community support, such as D3.js and FeatureIDE, should be considered. Using a 3D-rendering engine like Babylon.js allows to expand visualization concepts into a 3D shape, enabling a more comprehensive perspective.
- *If little programming knowledge is a requirement*, Google Charts or encapsulated D3.js components seem most useful.

Overall, we hope that our results and these conclusions help researchers as well as practitioners when designing novel techniques for engineering variant-rich systems that require visualizations.

6 THREATS TO VALIDITY

Selection bias is a potential threat to the validity of our study. We based our selection on two recent mapping studies and a manual search in the proceedings of three conferences, but we may still have missed relevant primary sources. Consequently, our study's findings may not represent the entire universe of research on visualizations for variant-rich system engineering. Still, our findings provide a valuable and detailed overview of the state-of-the-art.

Publication bias may have impacted our findings. We can only rely on published papers, which may not accurately represent the current state-of-the-art. Unpublished research, open-source tools, or industry practices may also be relevant and should be included in future research. Still, we capture a large number of primary sources and provide helpful insights for researchers and practitioners.

7 CONCLUSION

Visualizations are a powerful tool for understanding and analyzing software-engineering data, also in the context of variant-rich systems. We have examined 64 papers that have proposed visualizations for variant-rich systems and, building on 12 different (types of) tools: Eclipse EMF-GEF, Prefuse, D3.js, Graphviz, CCVisu, Google Charts, ConExp, Moose, Processing, FeatureIDE, Babylon.js, and Adhoc tools. To provide a detailed overview, we compared the visualization concepts used for the entities of variant-rich system engineering, the underlying ideas, the existing evidence on their usefulness, and the tools used. Our findings can help researchers and practitioners to understand what visualizations may be most useful to reuse, and to identify opportunities for exploring novel ones. We further provide help when selecting underlying tools for implementing a visualization. However, we want to emphasize that more systematic evaluations and comparisons are needed to better understand which visualization concepts are better suited for what entities of variant-rich system engineering. In future work, we aim to build on our findings to develop novel visualizations for under-explored concepts.

REFERENCES

[1] Jonas Åkesson, Sebastian Nilsson, Jacob Krüger, and Thorsten Berger. 2019. Migrating the Android Apo-Games into an Annotation-Based Software Product Line. In *SPLC*. ACM.

[2] Berima Andam, Andreas Burger, Thorsten Berger, and Michel R. V. Chaudron. 2017. FLOrIDA: Feature LOcatIon DAshboard for Extracting and Visualizing Feature Traces. In *VaMoS*.

[3] Nicolas Anquetil, Uirá Kulesza, Ralf Mitschke, Ana Moreira, Jean-Claude Royer, Andreas Rummler, and André Sousa. 2010. A Model-Driven Traceability Framework for Software Product Lines. *International Journal on Software & Systems Modeling* 9 (2010).

[4] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines*. Springer.

[5] Sven Apel and Dirk Beyer. 2011. Feature Cohesion in Software Product Lines: An Exploratory Study. In *ICSE*. IEEE.

[6] Mohsen Asadi, Samaneh Soltani, Dragan Gasevic, Marek Hatala, and Ebrahim Bagheri. 2014. Toward Automated Feature Model Configuration with Optimizing Non-Functional Requirements. *Information and Software Technology* 56, 9 (2014).

[7] Sana Ben Nasr, Guillaume Bécan, Mathieu Acher, João B. Ferreira Filho, Benoit Baudry, Nicolas Sannier, and Jean-Marc Davril. 2015. Matrixminer: A Red Pill to Architect Informal Product Descriptions in the Matrix. In *FSE*. ACM.

[8] Alexandre Bergel, Razan Ghzouli, Thorsten Berger, and Michel R. V. Chaudron. 2021. FeatureVista: Interactive Feature Visualization. In *SPLC*. ACM.

[9] Dirk Beyer. 2008. CCVisu: Automatic Visual Software Decomposition. In *ICSE-C*.

[10] Goetz Botterweck, Steffen Thiel, Ciarán Cawley, Daren Nestor, and André Preußner. 2008. Visual Configuration in Automotive Software Product Lines. In *COMPSAC*. IEEE.

[11] Goetz Botterweck, Steffen Thiel, Daren Nestor, Saad bin Abid, and Ciarán Cawley. 2008. Visual Tool Support for Configuring and Understanding Software Product Lines. In *SPLC*. IEEE.

[12] Ciarán Cawley, Goetz Botterweck, Patrick Healy, Saad B. Abid, and Steffen Thiel. 2009. A 3D Visualisation to Enhance Cognition in Software Product Line Engineering. In *ISVC*. Springer.

[13] Paul C. Clements and Charles W. Krueger. 2002. Point/Counterpoint: Being Proactive Pays Off / Eliminating the Adoption Barrier. *IEEE Software* 19, 4 (2002).

[14] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *VaMoS*. ACM.

[15] Thiago F. L. de Medeiros, Eduardo S. de Almeida, and Silvio R. de Lemos Meira. 2012. CodeScoping: A Source Code Based Tool to Software Product Lines Scoping. In *SEAA*. IEEE.

[16] Thiago H. B. de Oliveira, Martin Becker, and Elisa Y. Nakagawa. 2012. Supporting the Analysis of Bug Prevalence in Software Product Lines with Product Genealogy. In *SPLC*. ACM.

[17] Jamel Debbiche, Oskar Lignell, Jacob Krüger, and Thorsten Berger. 2019. Migrating Java-Based Apo-Games into a Composition-Based Software Product Line. In *SPLC*. ACM.

[18] Sybren Deelstra, Marco Sinnema, and Jan Bosch. 2005. Product Derivation in Software Product Families: A Case Study. *Journal of Systems and Software* 74, 2 (2005).

[19] Oscar Díaz, Raul Medeiros, and Leticia Montalvillo. 2019. Change Analysis of #if-def Blocks with FeatureCloud. In *SPLC*.

[20] Slawomir Duszynski and Martin Becker. 2012. Recovering Variability Information from the Source Code of Similar Software Products. In *PLEASE*. IEEE.

[21] Slawomir Duszynski, Jens Knodel, and Martin Becker. 2011. Analyzing the Source Code of Multiple Software Variants for Reuse Potential. In *WCRE*. IEEE.

[22] Slawomir Duszynski, Jens Knodel, Matthias Naab, Dirk Hein, and Clemens Schitter. 2008. Variant Comparison - A Technique for Visualizing Software Variants. In *WCRE*. IEEE.

[23] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. 2001. Graphviz—Open Source Graph Drawing Tools. In *GD*. Springer.

[24] Emelie Engström and Per Runeson. 2011. Software Product Line Testing - A Systematic Mapping Study. *Information and Software Technology* 53, 1 (2011).

[25] Sina Entekhabi, Anton Solback, Jan-Philipp Steghöfer, and Thorsten Berger. 2019. Visualization of Feature Locations with the Tool FeatureDashboard. In *SPLC*.

[26] Kevin Feichtinger, Daniel Hinterreiter, Lukas Linsbauer, Herbert Prähofer, and Paul Grünbacher. 2021. Guiding Feature Model Evolution by Lifting Code-Level Dependencies. *Journal of Computer Languages* 63 (2021).

[27] Alessio Ferrari, Giorgio O. Spagnolo, Stefania Gnesi, and Felice Dell'Orletta. 2015. CMT and FDE: Tools to Bridge the Gap Between Natural Language Documents and Feature Diagrams. In *SPLC*.

[28] Stefan Fischer, Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2014. Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants. In *ICSME*. IEEE.

[29] Thomas S. Fogdal, Helene Scherrebeck, Juha Kuusela, Martin Becker, and Bo Zhang. 2016. Ten Years of Product Line Engineering at Danfoss: Lessons Learned and Way Ahead. In *SPLC*. ACM.

[30] Willian M. Freire, Mamoru Massago, Arthur C. Zavadski, Aline M. Malachini, Miotto Amaral, and Thelma E. Colanzi. 2020. OPLA-Tool v2.0: A Tool for Product Line Architecture Design Optimization. In *SBES*.

[31] Muhammad Garba, Adel Noureddine, and Rabih Bashroush. 2016. Musa: A Scalable Multi-Touch and Multi-Perspective Variability Management Tool. In *WICSA*. IEEE.

[32] Gharib Gharibi and Yongjie Zheng. 2016. ArchFeature: Integrating Features into Product Line Architecture. In *SAC*.

[33] Jeffrey Heer, Stuart K. Card, and James A. Landay. 2005. Prefuse: A Toolkit for Interactive Information Visualization. In *CHI*. ACM.

[34] Florian Heidenreich, Ilie Savga, and Christian Wende. 2008. On Controlled Visualisations in Software Product Line Engineering. In *SPLC*.

[35] André Heuer, Kim Lauenroth, Marco Müller, and Jan-Nils Scheele. 2010. Towards Effective Visual Modeling of Complex Software Product Lines. In *SPLC*.

[36] Daniel Hinterreiter, Paul Grünbacher, and Herbert Prähofer. 2020. Visualizing Feature-Level Evolution in Product Lines: A Research Preview. In *REFSQ*. Springer.

[37] Jose-Miguel Horcas, Jose A. Galindo, and David Benavides. 2022. Variability in Data Visualization: A Software Product Line Approach. In *SPLC*.

[38] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2019. Software Product Line Engineering: A Practical Experience. In *SPLC*.

[39] Tom Huysegoms, Monique Snoeck, Guido Dedene, Antoon Goderis, and Frank Stumpe. 2013. Visualizing Variability Management in Requirements Engineering through Formal Concept Analysis. *Procedia Technology* 9 (2013).

[40] Sheny Illescas, Roberto E. Lopez-Herrejon, and Alexander Egyed. 2016. Towards Visualization of Feature Interactions in Software Product Lines. In *VISSOFT*. IEEE.

[41] Aleksandar Jakšić, Robert B. France, Philippe Collet, and Sudipto Ghosh. 2014. Evaluating the Usability of a Visual Feature Modeling Notation. In *SLE*. Springer.

[42] Tetsuya Kanda, Takashi Ishio, and Katsuro Inoue. 2013. Extraction of Product Evolution Tree from Source Code of Product Variants. In *SPLC*.

[43] Tetsuya Kanda, Takashi Ishio, and Katsuro Inoue. 2015. Approximating the Evolution History of Software from Source Code. *IEICE Transactions on Information and Systems* 98, 6 (2015).

[44] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. Carnegie Mellon University.

[45] Christian Kästner, Salvador Trujillo, and Sven Apel. 2008. Visualizing Software Product Line Variabilities in Source Code. In *SPLC*.

[46] Manjinder Kaur and Parveen Kumar. 2014. Spotting the Phenomenon of Bad Smells in MobileMedia Product Line Architecture. In *IC3*. IEEE.

[47] Youngtaek Kim, Hyeon Jeon, Young-Ho Kim, Yuhoon Ki, Hyunjoo Song, and Jinwook Seo. 2021. Visualization Support for Multi-Criteria Decision Making in Software Issue Propagation. In *PacificVis*. IEEE.

[48] Sebastian Krieter, Marcus Pinnecke, Jacob Krüger, Joshua Sprey, Christopher Sontag, Thomas Thüm, Thomas Leich, and Gunter Saake. 2017. FeatureIDE: Empowering Third-Party Developers. In *SPLC*. ACM.

[49] Jacob Krüger. 2021. *Understanding the Re-Engineering of Variant-Rich Systems: An Empirical Work on Economics, Knowledge, Traceability, and Practices.* Ph. D. Dissertation. Otto-von-Guericke University Magdeburg.
[50] Jacob Krüger and Thorsten Berger. 2020. Activities and Costs of Re-Engineering Cloned Variants Into an Integrated Platform. In *VaMoS*. ACM.
[51] Jacob Krüger and Thorsten Berger. 2020. An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse. In *ESEC/FSE*. ACM.
[52] Jacob Krüger, Wolfram Fenske, Thomas Thüm, Dirk Aporius, Gunter Saake, and Thomas Leich. 2018. Apo-Games - A Case Study for Reverse Engineering Variability from Cloned Java Variants. In *SPLC*. ACM.
[53] Jacob Krüger, Wardah Mahmood, and Thorsten Berger. 2020. Promote-pl: A Round-Trip Engineering Process Model for Adopting and Evolving Product Lines. In *SPLC*. ACM.
[54] Jacob Krüger, Louis Nell, Wolfram Fenske, Gunter Saake, and Thomas Leich. 2017. Finding Lost Features in Cloned Systems. In *SPLC*.
[55] Elias Kuiter, Jacob Krüger, Sebastian Krieter, Thomas Leich, and Gunter Saake. 2018. Getting Rid of Clone-And-Own: Moving to a Software Product Line for Temperature Monitoring. In *SPLC*. ACM.
[56] Elias Kuiter, Jacob Krüger, and Gunter Saake. 2021. Iterative Development and Changing Requirements: Drivers of Variability in an Industrial System for Veterinary Anesthesia. In *SPLC*. ACM.
[57] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. 2010. An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines. In *ICSE*. ACM.
[58] Robert Lindohf, Jacob Krüger, Erik Herzog, and Thorsten Berger. 2021. Software Product-Line Evaluation in the Large. *Empirical Software Engineering* 26, 30 (2021).
[59] Felix Loesch and Erhard Ploedereder. 2007. Optimization of Variability in Software Product Lines. In *SPLC*. ACM.
[60] Roberto E Lopez-Herrejon and Alexander Egyed. 2013. Towards Interactive Visualization Support for Pairwise Testing Software Product Lines. In *VISSOFT*. IEEE.
[61] Roberto Erick Lopez-Herrejon, Sheny Illescas, and Alexander Egyed. 2018. A Systematic Mapping Study of Information Visualization for Software Product Line Engineering. *Journal of Software: Evolution and Process* 30, 2 (2018).
[62] Mike Mannion and David Sellier. 2007. Visualising Product Line Requirement Selection Decisions.
[63] Jabier Martinez, Xhevahire Tërnava, and Tewfik Ziadi. 2018. Software Product Line Extraction from Variability-Rich Systems: The Robocode Case Study. In *SPLC*.
[64] Jabier Martinez and Anil Kumar Thurimella. 2012. Collaboration and Source Code Driven Bottom-Up Product Line Engineering. In *SPLC*.
[65] Jabier Martinez, Tewfik Ziadi, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Name Suggestions during Feature Identification: The VariClouds Approach. In *SPLC*.
[66] Jabier Martinez, Tewfik Ziadi, Jacques Klein, and Yves Le Traon. 2014. Identifying and Visualising Commonality and Variability in Model Variants. In *ECMFA*. Springer.
[67] Jabier Martinez, Tewfik Ziadi, Raul Mazo, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2014. Feature Relations Graphs: A Visualisation Paradigm for Feature Constraints in Software Product Lines. In *VISSOFT*. IEEE.
[68] Raul Medeiros, Jabier Martinez, Oscar Díaz, and Jean-Rémy Falleri. 2022. Visualizations for the Evolution of Variant-Rich Systems: A Systematic Mapping Study. *Information and Software Technology* (2022).
[69] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer.
[70] Leticia Montalvillo, Oscar Díaz, and Maider Azanza. 2017. Visualizing Product Customization Efforts for Spotting SPL Reuse Opportunities. In *SPLC*.
[71] Leticia Montalvillo, Oscar Díaz, and Thomas Fogdal. 2018. Reducing Coordination Overhead in SPLs: Peering in on Peers. In *SPLC*.
[72] Johann Mortara, Philippe Collet, and Anne-Marie Dery-Pinna. 2021. Visualization of Object-Oriented Variability Implementations as Cities. In *VISSOFT*. IEEE.
[73] Johann Mortara, Philippe Collet, and Anne-Marie Pinna-Dery. 2022. IDE-Assisted Visualization of Indebted OO Variability Implementations. In *SPLC*. ACM.
[74] Alexandr Murashkin, Michał Antkiewicz, Derek Rayside, and Krzysztof Czarnecki. 2013. Visualization and Exploration of Optimal Variants in Product Line Engineering. In *SPLC*. ACM.
[75] Damir Nešić, Jacob Krüger, Ștefan Stănciulescu, and Thorsten Berger. 2019. Principles of Feature Modeling. In *ESEC/FSE*. ACM.
[76] Daren Nestor, Steffen Thiel, Goetz Botterweck, Ciarán Cawley, and Patrick Healy. 2008. Applying Visualisation Techniques in Software Product Lines. In *SoftVis*.
[77] Oscar Nierstrasz, Stéphane Ducasse, and Tudor Gîrba. 2005. The Story of Moose: An Agile Reengineering Environment. In *ESEC/FSE*.
[78] Nan Niu and Steve Easterbrook. 2009. Concept Analysis for Product Line Requirements. In *AOSD*.
[79] Andy J. Nolan and Silvia Abrahão. 2010. Dealing with Cost Estimation in Software Product Lines: Experiences and Future Directions. In *SPLC*. Springer.
[80] Christopher Pietsch, Timo Kehrer, Udo Kelter, Dennis Reuling, and Manuel Ohrndorf. 2015. SiPL–A Delta-Based Modeling Framework for Software Product Line Engineering. In *ASE*. IEEE.
[81] Andreas Pleuss and Goetz Botterweck. 2012. Visualization of Variability and Configuration Options. *International Journal on Software Tools for Technology Transfer* 14 (2012).
[82] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering*. Springer.
[83] Rick Rabiser, Deepak Dhungana, Wolfgang Heider, and Paul Grünbacher. 2009. Flexibility and End-User Support in Model-Based Product Line Tools. In *SEAA*. IEEE.
[84] Márcio Ribeiro, Társis Tolêdo, Johnni Winther, Claus Brabrand, and Paulo Borba. 2012. Emergo: A Tool for Improving Maintainability of Preprocessor-Based Product Lines. In *AOSD*.
[85] Dan Rubel, Jaime Wren, and Eric Clayberg. 2011. *The Eclipse Graphical Editing Framework (GEF)*. Addison-Wesley.
[86] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. 2015. Cloned Product Variants: From Ad-Hoc to Managed Software Product Lines. *International Journal on Software Tools for Technology Transfer* (2015).
[87] Alcemir R. Santos, Ivan d. C. Machado, and Eduardo S. de Almeida. 2016. RiPLE-HC: Visual Support for Features Scattering and Interactions. In *SPLC*.
[88] Alexander Schlie, Kamil Rosiak, Oliver Urbaniak, Ina Schaefer, and Birgit Vogel-Heuser. 2019. Analyzing Variability in Automation Software with the Variability Analysis Toolkit. In *SPLC*.
[89] Klaus Schmid and Martin Verlage. 2002. The Economic Impact of Product Line Adoption and Evolution. *IEEE Software* 19, 4 (2002).
[90] David Sellier and Mike Mannion. 2007. Visualising Product Line Requirement Selection Decision Inter-Dependencies. In *REV*. IEEE.
[91] Ștefan Stănciulescu, Sandro Schulze, and Andrzej Wąsowski. 2015. Forked and Integrated Variants in an Open-Source Firmware Project. In *ICSME*. IEEE.
[92] Michael Stengel, Mathias Frisch, Sven Apel, Janet Feigenspan, Christian Kästner, and Raimund Dachselt. 2011. View Infinity: A Zoomable Interface for Feature-Oriented Software Development. In *ICSE*.
[93] Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, and Thorsten Berger. 2019. Facing the Truth: Benchmarking the Techniques for the Evolution of Variant-Rich Systems. In *SPLC*. ACM.
[94] Reinhard Tartler, Daniel Lohmann, Julio Sincero, and Wolfgang Schröder-Preikschat. 2011. Feature Consistency in Compile-Time-Configurable System Software: Facing the Linux 10,000 Feature Problem. In *EuroSys*. ACM.
[95] Xhevahire Tërnava, Johann Mortara, and Philippe Collet. 2019. Identifying and Visualizing Variability in Object-Oriented Variability-Rich Systems. In *SPLC*.
[96] Nishanth Thimmegowda and Jörg Kienzle. 2015. Visualization Algorithms for Feature Models in Concern-Driven Software Development. In *MODULARITY*.
[97] Thomas Thüm, Sebastian Krieter, and Ina Schaefer. 2018. Product Configuration in the Wild: Strategies for Conflicting Decisions in Web Configurators. In *ConfWS*. CEUR-WS.org.
[98] Pablo Trinidad, Antonio R. Cortés, David Benavides, and Sergio Segura. 2008. Three-Dimensional Feature Diagrams Visualization. In *SPLC*.
[99] Simon Urli, Alexandre Bergel, Mireille Blay-Fornarino, Philippe Collet, and Sébastien Mosser. 2015. A Visual Support for Decomposing Complex Feature Models. In *VISSOFT*. IEEE.
[100] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software Product Lines in Action*. Springer.
[101] Krzysztof Wnuk, Björn Regnell, and Lena Karlsson. 2009. What Happened to Our Features? Visualization and Understanding of Scope Change Dynamics in a Large-Scale Industrial Setting. In *RE*. IEEE.
[102] Serhiy A Yevtushenko. 2000. System of Data Analysis "Concept Explorer". In *Proc. 7th National Conference on Artificial Intelligence (KII'00)*.
[103] Kentaro Yoshimura, Dharmalingam Ganesan, and Dirk Muthig. 2006. Defining a Strategy to Introduce a Software Product Line Using Existing Embedded Systems. In *SPLC*. ACM.
[104] Anna Zamansky and Iris Reinhartz-Berger. 2017. Visualizing Code Variabilities for Supporting Reuse Decisions. In *SCME-iStarT@ER*.
[105] Nick Q. Zhu. 2013. *Data Visualization with D3.js Cookbook*. Packt Publishing.