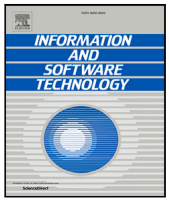
Information and Software Technology 154 (2023) 107084

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Visualizations for the evolution of Variant-Rich Systems: A systematic mapping study

Raul Medeiros [a,*], Jabier Martinez [b], Oscar Díaz [a], Jean-Rémy Falleri [c]

[a] University of the Basque Country (UPV/EHU), San Sebastián, Spain
[b] Tecnalia, Basque Research and Technology Alliance (BRTA), Derio, Spain
[c] LaBRI, UMR 5800, F-33400, IUF, Talence, France

ARTICLE INFO

Keywords:
Variant-rich systems
Software product lines
Visualization
Evolution
Maintenance
Mapping study

ABSTRACT

Context: Variant-Rich Systems (VRSs), such as Software Product Lines or variants created through clone & own, aim at reusing existing assets. The long lifespan of families of variants, and the scale of both the code base and the workforce make VRS maintenance and evolution a challenge. Visualization tools are a needed companion.

Objective: We aim at mapping the current state of visualization interventions in the area of VRS evolution. We tackle evolution in both functionality and architecture. Three research questions are posed: What sort of analysis is being conducted to assess VRS evolution? (Analysis perspective); What sort of visualizations are displayed? (Visualization perspective); What is the research maturity of the reported interventions? (Maturity perspective).

Methods: We performed a systematic mapping study including automated search in digital libraries, expert knowledge, and snowballing.

Results: The study reports on 41 visualization approaches to cope with VRS evolution. Analysis wise, feature identification and location is the most popular scenario, followed by variant integration towards a Software Product Line. As for visualization, nodelink diagram visualization is predominant while researchers have come up with a wealth of ingenious visualization approaches. Finally, maturity wise, almost half of the studies are solution proposals. Most of the studies provide proof-of-concept, some of them also include publicly available tools, yet very few face proof-of-value.

Conclusions: This study introduces a comparison framework where to frame future studies. It also points out distinct research gaps worth investigating as well as shortcomings in the evidence about relevance and contextual considerations (e.g., scalability).

1. Introduction

Software evolution is a major issue in software engineering [1]. Its understanding involves facing large amounts of data from heterogeneous data sources including version control systems, bug tracking systems, mailing and project discussion lists, as well as the implementation assets themselves. Software evolution research interests include to build theories and models that allow grasping the past and present, while predicting future properties, and hence, supporting software maintenance and evolution [1]. In this realm, our endeavours can be framed along two dimensions: the purpose of the evolution (i.e., evolution in functionality, e.g., perfective maintenance, or evolution in architecture, e.g., refactoring) and the subject of evolution (i.e., variant-rich systems).

A Variant-Rich System (VRS) consists of a family of software products (aka portfolio) that serves a particular market segment or context while sharing commonalities and variabilities among them in their functions or behavior. Depending on the automation level (*ad-hoc* vs. systematic), there are different strategies for managing VRSs, ranging from clone & own to Software Product Lines (SPLs) [2]. VRS evolution tends to be more frequent and complex than its one-off application counterpart. Frequency wise, evolution episodes are more numerous since the functionality scope is not that of a single product but of a family of products (aka a domain). On top of that, evolution is not limited to the functionality itself. The gradual enlargement of the functionality might require to be accompanied by the evolution of the system architecture (e.g., moving from clone & own to SPLs [3]). Both

* Corresponding author.
E-mail addresses: raul.medeiros@ehu.eus (R. Medeiros), jabier.martinez@tecnalia.com (J. Martinez), oscar.diaz@ehu.eus (O. Díaz), falleri@labri.fr (J.-R. Falleri).

https://doi.org/10.1016/j.infsof.2022.107084
Received 14 January 2022; Received in revised form 18 September 2022; Accepted 22 September 2022
Available online 29 September 2022

higher frequency and more complex evolution scenarios substantiate the need for visualization assistance even to a larger degree than in one-off software development, and, specifically, the use of Visual Analytics [4].

Visual Analytics is defined as the science of analytical reasoning facilitated by interactive visual interfaces [4]. The basic idea is to visually represent the data to allow humans to directly interact with the information, to gain insights, and to ultimately make optimal decisions. Broadly speaking, Visual Analytics helps create a path from data to decision. In this process, visualization plays a twofold role [5]:

- as an abstraction medium, by highlighting certain constructs and relationships while ignoring others,
- as an interaction medium, by supporting the workflows for decision making.

Hence, interventions based on this approach are characterized in terms of both *the analysis* they allow, and *the visualizations* they support. On these premises, this work introduces the following research questions:

- **RQ1:** *What sort of analysis is being conducted to assess VRS evolution?*
- **RQ2:** *What sort of visualizations are displayed?*
- **RQ3:** *What is the research maturity of the reported interventions?*

We answer these questions through a systematic mapping study. A systematic mapping study is an evidence-based form of secondary study. Systematic mapping studies intend to provide a broad overview of a study domain, as well as to determine whether or not research evidence exists on a certain issue and to estimate its amount [6]. To the best of our knowledge, no previous work has characterized VRS literature along with the aforementioned research questions. We extracted data for 234 potentially relevant articles using academic article search engines. After a careful screening of these articles, we identified 41 primary sources.[1]

The rest of the paper is structured as follows. Section 2 presents related reviews. Section 3 describes the background. Section 4 introduces the research design, and Section 5 describes the data extraction. Sections 6, 7 and 8 present the results while Section 9 discusses them. Then, Section 10 presents the threats to validity. Finally, Section 11 concludes the paper.

2. Related work

Several studies have surveyed the literature on software visualization, namely:

- Kienle and Müller look into requirements to build and evaluate tools in software visualization along with three main activities: maintenance, re-engineering, and reverse engineering [7].
- Shahin et al. investigated software visualizations but for understanding software architectures [8].
- Chotisarn et al. conduct a systematic review along two main concerns: software aspect to be visualized (i.e., structure, behavior and evolution) and software process to be supported (i.e., software requirements, software design and implementation, software validation, software maintenance) [9].
- Novais et al. review the visualization of software evolution from a global perspective [10].
- Lopez-Herrejon et al. focus on visualization approaches to support a broad range of Software Product Line activities [11].

Table A.15 provides more details on the mentioned surveys. In their tertiary study on research in Software Product Lines (SPLs) [12], Raatikainen et al. notice that [11] and its previous version [13], are the only ones which tackle visualization in SPLs, departing from visualization in one-off development. Compared with Lopez-Herrejon et al.'s work, our mapping is more specific as we focus on evolution. Yet, our interest is broader since we encompass VRSs no matter their stage in the evolution maturity journey (e.g., clone & own) and not only SPLs. This introduces a main distinctive concern that is not limited to the 'evolution in functionality' but also expands to 'evolution in architecture'. This results in [11] considering 37 primary sources while ours had 41 with an overlap of 12 primary sources.

3. The mapping space

Visual Analytics is defined as the science of analytical reasoning facilitated by interactive visual interfaces [4]. Accordingly, primary sources are characterized along two major dimensions: the analysis dimension and the visualization dimension. We add a third dimension: the maturity of the visualization proposal. Along with good practices in mapping studies, this section resorts to literature-backed classification schemas that will be used to characterize the primary sources along these three dimensions.

3.1. Dimension 1: Analysis scenarios

This analysis dimension considers the subject (aka 'object', 'artifact') to be analyzed, and the purpose of the analysis.

The subject of evolution. VRSs face variability at different degrees of automation (*ad-hoc* vs. systematic) through different implementation mechanisms: annotative approaches like pre-processor directives [14] or feature toggles [15]; compositional approaches such as source code super-imposition and merge [16]; generative programming [17], feature-oriented programming [18,19], aspect-oriented programming [20] or delta-oriented programming [21]. The binding time of the variability is also relevant ranging from design-time to runtime, the latter enabling the creation of self-adaptive systems (aka Dynamic SPLs [22]). We use the umbrella term 'VRSs' to denote this broad diversity of cases, from almost independently created variants (i.e., clone & own), to highly-configurable systems, SPLs and Dynamic SPLs.

The purpose of evolution. VRSs can evolve not only in their functionality but also in their architecture. At the onset, an organization might capture variations through 'if-then' statements, where 'if' conditions are evaluated at runtime against a configuration file. Here, a single artifact handles all functionality, no matter whether it is statically known that a configuration option will never be selected in a certain scenario. Alternatively, variations can be supported as distinct variants (products). While the number of variants is small, clone & own might be an effective architecture. However, as the variants increase, clone & own ends in replication and propagation challenges [2]. Engineers can opt to move to *Conditional Compilation*. Here, variants are enclosed within *#ifdef* and *#endif* marks, and associated with pre-compilation directives, i.e., boolean expressions upon 'configuration parameters'. This allows for configuration-dependency checking to be outsourced from the application code to dedicated configurators. Finally, as the number of features and variants increase, a more systematic approach is needed. This might end up in an SPL.

This diversity in both the subject and the purpose gives rise to distinct scenarios where engineers are confronted with VRS evolution. Next, we revise Struber et al.'s evolution scenarios [23]. For each scenario, we present: the setting where this scenario arises and an example of a visualization intervention that (partially) tackles these needs. We selected the examples arbitrarily based on visualization variety, maturity, and clarity.

[1] A gallery of the visualizations of the 41 primary sources (taken from the respective papers) is available for research purposes at https://doi.org/10.5281/zenodo.7057654.

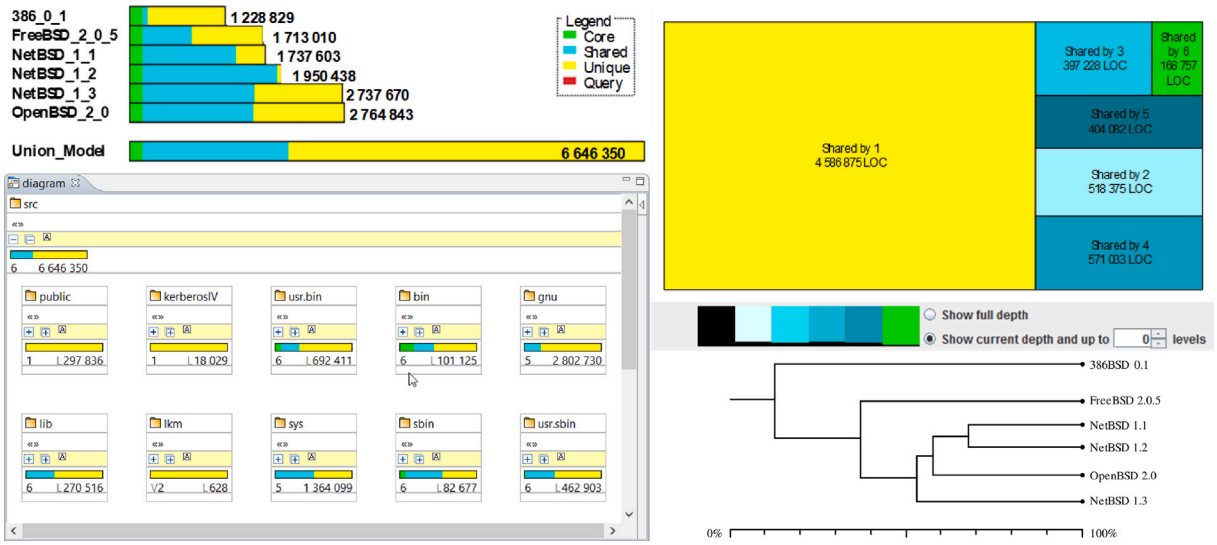


Fig. 1. *Variant Synchronization.* Using similarity analysis, this tool identifies common, shared and unique artifacts among variants. On the upper left side, the visualization shows a similarity distribution of 6 variants. Colors represent the degree of commonality of the artifacts: green (common), cyan (shared) and yellow (unique). On the upper right side, a treemap groups the shared lines of code regarding the number of products where they are present. On the bottom left side, the similarity distribution is displayed within each implementation element (i.e., package or folder) allowing a more focused structural exploration. Finally, on the bottom right corner, a phylogenetic distance diagram provides information about which variants are more similar. Figure credits [S1]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.1.1. Variant Synchronization (VS)

Setting. Clone & own and variant-based evolved SPLs. When bug fixes or feature implementations are made in one variant, they need to be propagated to other variants. During Variant Synchronization, it is important to identify the variants affected by a bug fix or a feature upgrade. To this end, criteria such as the cost of the propagation or the importance of the variant can be used.

Example. Variant synchronization benefits from visualizing the distribution of the lines of code of each variant in terms of 'core', 'shared', or 'unique'. In this way, if a bug is fixed in the 'shared' part of a variant it becomes a candidate to be rippled across the other variants sharing the same code. This approach is followed in [S1] using set-based similarity analysis (see Fig. 1).

3.1.2. Variant Integration (VI)

Setting. Variant-based evolved SPLs. Here, application engineers are allowed to tune the codebase to meet the specifics of a variant without waiting for this specific to be introduced in the SPL platform. Bug fixes or urgent customer requests might not wait for being conducted at the SPL platform, and hence, they are branched off into a separate codebase. Eventually, these branches need to be merged back into the SPL Master. Being variants from the same SPL, they are based on the very same core assets, and hence, bug fixes or functional enhancements undertaken for a variant might well serve other variants. Here, application engineers need to be aware of potential coordination problems right during coding rather than deferring it until merging time.

Example. *PeeringHub* [S2] (see Fig. 2) advocates for making application engineers aware of potential coordination problems right during coding rather than deferring it until merge time. To this end, *PeeringHub* introduces the notion of 'peering bar' for GitHub. These visual bars reflect whether the codebase of a variant's feature is being upgraded in other variant branches. *PeeringHub* helps by making developers aware of potential coordination problems right during variant coding rather than deferring it until variant specifics are merged back into the platform *master*.

3.1.3. Feature Identification and Location (FIL)

Setting. Migration from clone & own to an SPL platform. Feature identification aims to determine which features are present, while feature location tackles the relationship between features and assets. Developers may wish to determine which features already exist in the system and which features are implemented in which assets (e.g., source code, models, requirements, or other types of artifacts) as a means of better supporting clone & own development as well as preparing for the migration to an SPL platform.

Example. During the migration from clone & own to an SPL platform, SPL engineers might need to identify which parts of the variants are common and which ones are specific. Fig. 3 shows a visualization that uses clone detection for this purpose. The visualization shows the parts of the source code that are common (red), variant specific (white), and those that might be shared among some variants but not all (light red). This way, the user can interact with the results (e.g., focusing on specific source code files) and identify mandatory features and features that only appear in a given variant or a set of variants [S3].

3.1.4. Constraints Extraction (CE)

Setting. Migration from clone & own to an SPL platform. On the way towards an SPL, meaningful sets of functional requirements are named as a 'feature'. Features hold dependencies among them: a feature might require/exclude other features. This information might not be initially known. Developers might be assisted through an automated constraint extraction approach. Constraint extraction exhibits two needs. First, it has to be easy to identify which constraints are already formalized. Secondly, constraints extraction techniques are not 100% accurate, and hence, means are needed for measuring the confidence of the extracted constraints.

Example. *Feature Relation Graphs* (*FRoGs*) [S4]. Fig. 4 places the feature under analysis at the center while the rest of the features are disposed at a distance based on the frequency they appear together in the available configurations. It can be used to discover non-formalized 'requires' and 'excludes' constraints between features as well as soft-constraints such as one feature 'encourages' or 'discourages' another.

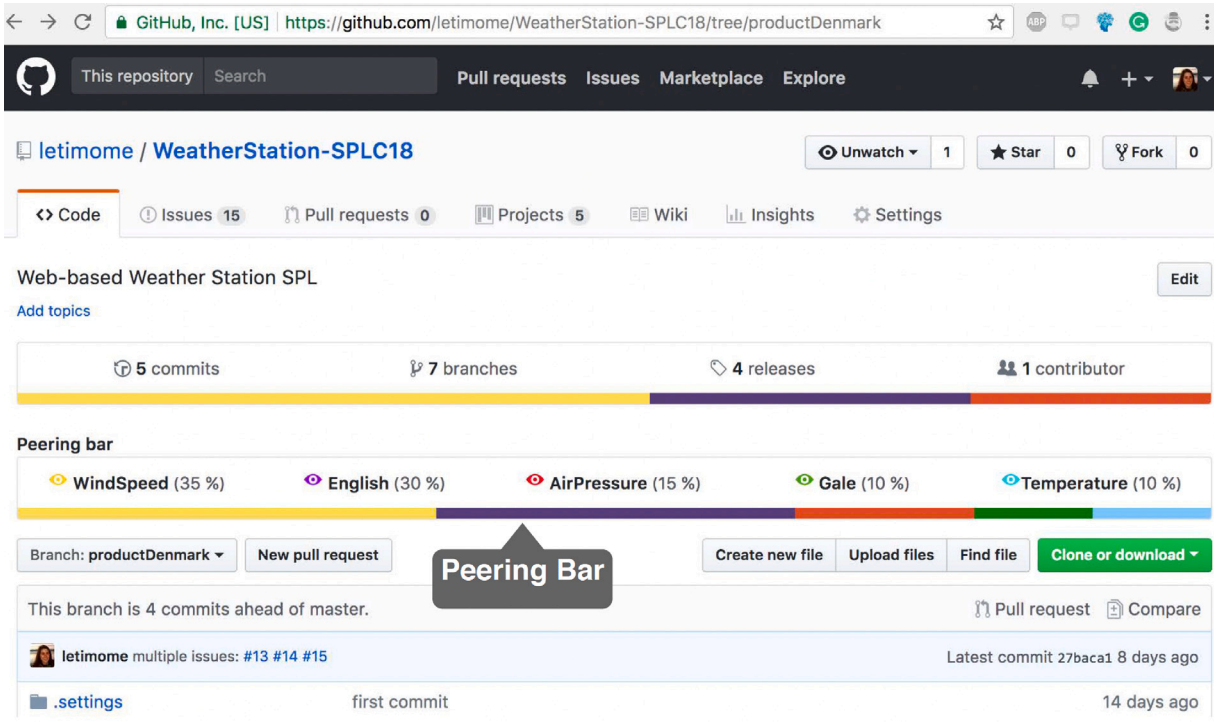


Fig. 2. *Variant Integration. PeeringHub* mimics GitHub topic bar but now colors denote code churns per feature within a branch. Branches hold temporary developments of variants (e.g., *productDenmark*) before being merged back into the Master. In the example, the *PeeringBar* hints at code efforts being conducted for the variant *productDenmark*, specifically, for features *WindSpeed*, *English*, *AirPressure*, etc. In this way, if you are upgrading namesake features, you better engage with your *productDenmark* colleagues to prevent later conflicts during merging. Figure credits [S2]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

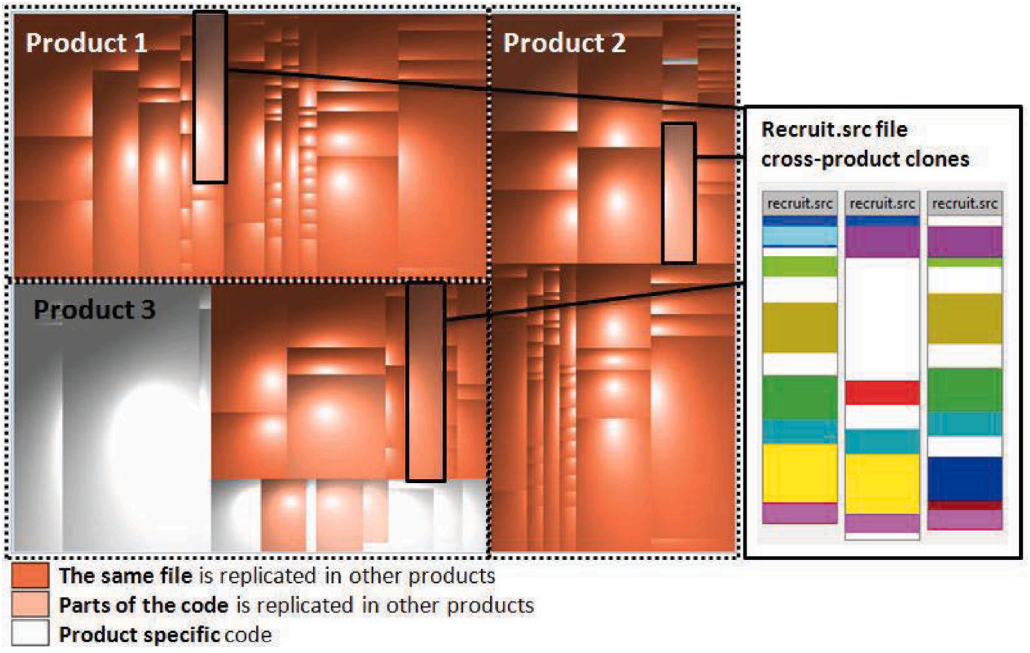


Fig. 3. *Feature Identification and Location.* The visualization consists of a heatmap inside a treemap. The treemap represents the structure of each variant's artifacts, while the heatmap represents the degree of commonality of each variant implementation artifact. Using the colors, a developer can identify the parts of the source code that are common (red), the parts that are product specific (white), and the parts that might be shared among some product variants but not by all of them (light red). Figure credits [S3]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

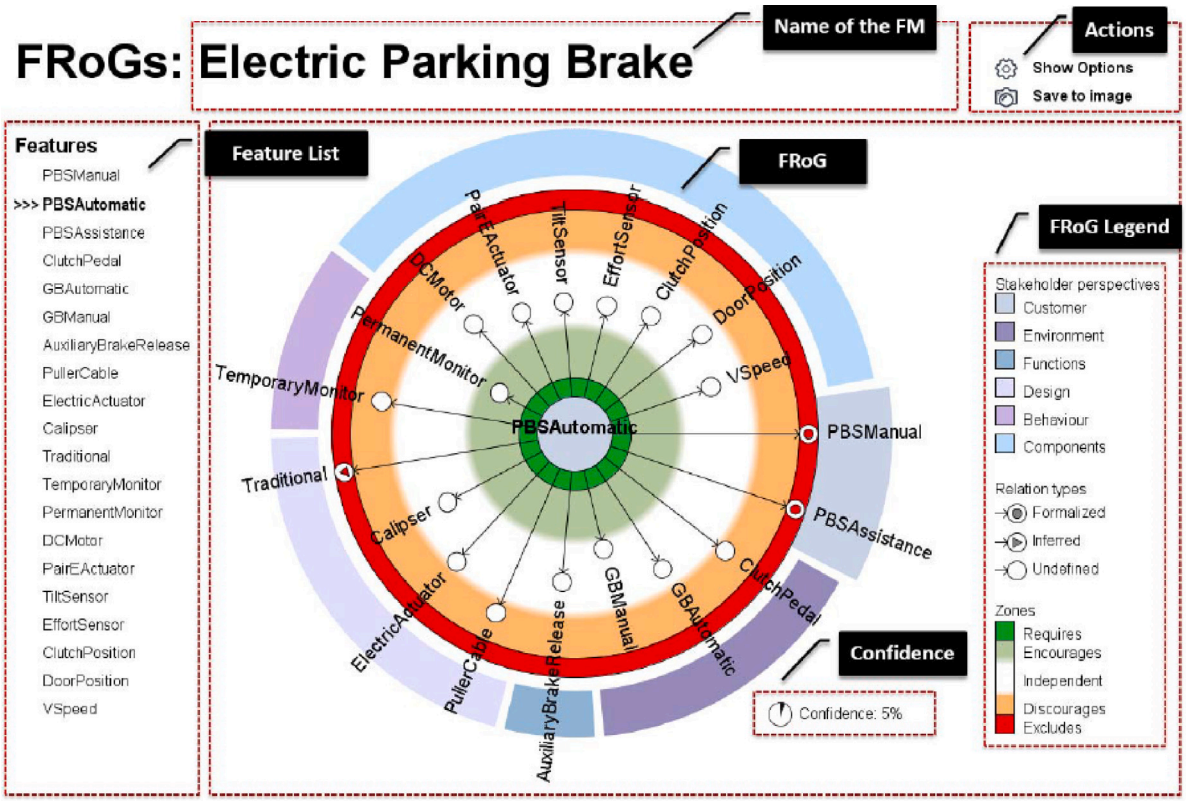


Fig. 4. *Constraint Extraction. Feature Relation Graphs* use nodelink diagrams on top of a radial network. Nodes represent feature names. The node in the center is the feature under analysis while the arrangement of the others represents the relationship they have with the central feature. Concretely, the arrangement represents three types of relationships: type of constraint (proximity to the center), stakeholder perspective (radial position), and level of formalization (edge type). Using the visualization, developers identify potential feature constraints related to the feature under analysis. Figure credits [S4]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

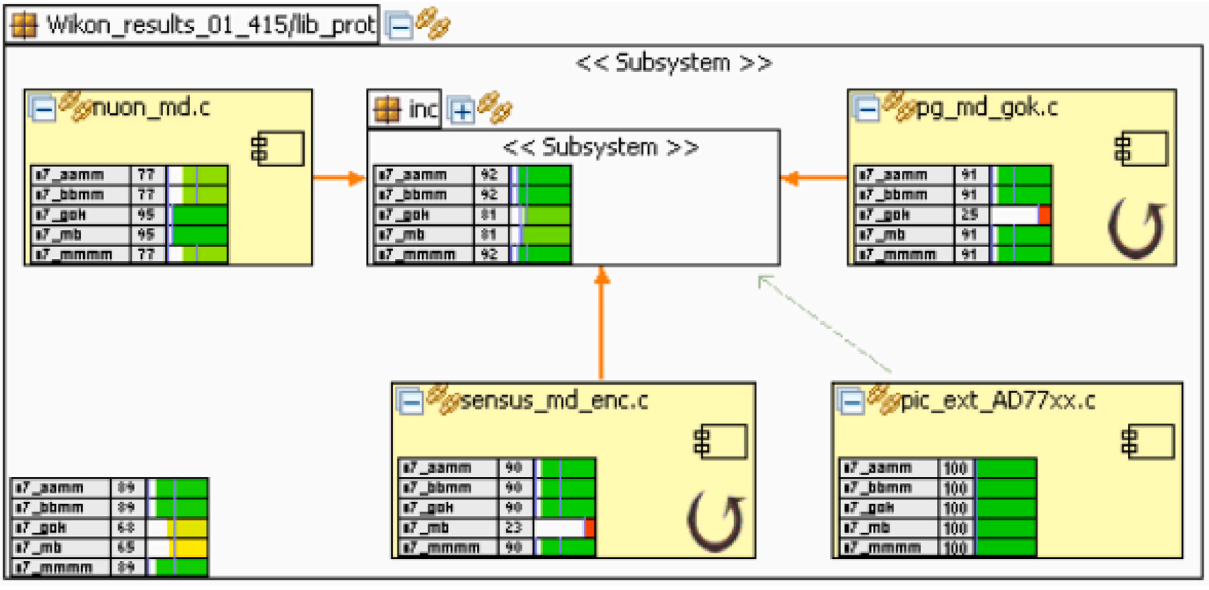


Fig. 5. *Architecture Recovery*. When it comes to integrating a set of existing similar variants on the way towards an SPL, it is essential to abstract a common architecture. This visualization abstracts from the variants' component architecture to highlight the variance presence. The component diagram depicts which modules are present in several variants (list of names) and to which extent the module is similarly implemented (bar chart). Figure credits [S5].

3.1.5. Architecture Recovery (AR)

Setting. Migration from clone & own to an SPL platform. While feature models capture functionality dependencies, architectural models focus on the dependencies of implemented classes or components. Automatic extraction of architectural dependencies can be assisted through architecture recovery techniques. Architecture Recovery can be performed from a configurable platform or from a set of variants.

Example. Building a platform architecture out of existing variants is error-prone and tedious. Fig. 5 introduces a visualization that abstracts and aggregates this information. Commonalities and variabilities

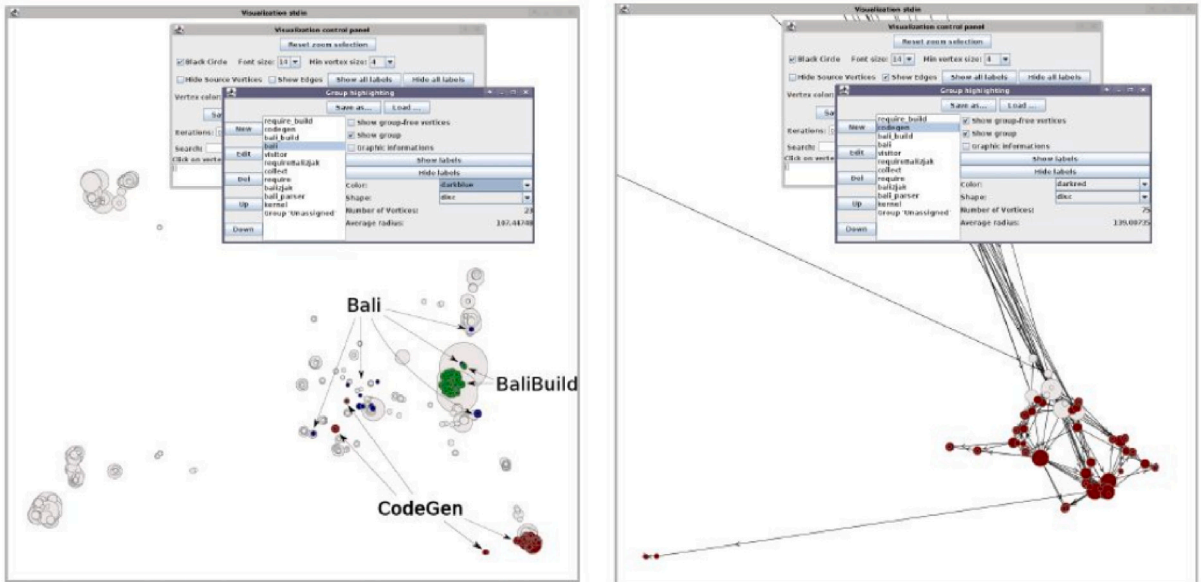


Fig. 6. *Transformation. FeatureVisu* uses a nodelink visualization built on top of a clustering method to depict the feature cohesion metric. For instance, on the left side, we can observe how the implementation elements of the feature BaliBuild (green) are more cohesive than those of the features Bali (blue) and CodeGen (red). On the right side, we can observe an excerpt of the previous layout but this time showing the dependency links between the implementation elements. Figure credits [S6]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

among variant architectures are captured from two perspectives: variance presence (i.e., if the existence of the module is variable or not) and variance realization (i.e., if the implementation of the module is different across variants) [S5].

3.1.6. Transformations (TR)

Setting. Evolution of an SPL platform. The developer can use transformation techniques to decrease manual effort during evolution activities. Transformation techniques include refactoring and partial refactoring of an SPL platform. For instance, high cohesion of a feature is a desired property and a relevant motivation for refactoring. For this, developers look to what extent implementation elements of a feature depend on each other, minimizing external dependencies.

Example. *FeatureVisu* (see Fig. 6) supports feature cohesion by creating clustering among features with a high cohesion.

3.1.7. Functional Testing (FT)

Setting. Clone & own and SPL platforms. Regression testing aims at ensuring that an application still functions as expected after any code changes. In view of the impossibility of testing hundreds of features and variants, deciding which features and configuration to test becomes critical. To tame this scalability issue, testers resort to distinct heuristics: hot spots (i.e., the larger the change, the more likely the impact) or variant similarity comparison analysis to increase test coverage.

Example. *FeatureCloud* (see Fig. 7) is a tree-cloud visualization to identify hot-spot features, i.e., features undergoing substantial change in their codebase, i.e., *ifdef* blocks. *FeatureCloud* depicts feature change along three metrics: LOC (color opacity), scattering (font size), and tangling (position in the word cloud). Using *FeatureCloud*, developers identify which features had complex updates and thus, might require more dedicated testing.

3.1.8. Analysis of Non-Functional Properties (ANF)

Setting. Clone & own and SPL platforms. VRSs are common among cyber–physical systems where non-functional requirements are key (e.g., performance in a safety–critical system, memory consumption in an embedded system with resource constraints). In safety–critical domains, engineers might need to explore which variants are optimal for their goals. This might require the analysis of the impact of features and feature interactions on quality properties, optimization of configurations towards attaining a given quality criteria, and analysis of trade-offs among non-functional properties.

Example. Fig. 8 presents a bubble chart to analyze optimal variants: variants are located based on four custom non-functional metrics (battery-life, security, productivity and cost).

3.1.9. Co-evolution of Problem Space and Solution Space (CPS)

Setting. Compared to the evolution of one-off applications, VRS evolution introduces additional challenges since the evolution should be synchronized across a potentially high number of variants [2]. The inter-dependency between these variants varies from highly-independent (e.g., the clone & own approach where 'clones' might evolve totally independently from its ancestor) to totally-dependent variants (e.g., configurable software-product lines where new variant requirements should be first considered at the core platform before impacting the variants). An automated approach may suggest co-evolution activities to maintain the two spaces in sync. This scenario includes both evolving the solution space based on the problem space, and vice-versa.

Example. New feature dependencies may be reflected uniquely in the code without being mirrored in the feature model due to development urgency during SPL evolution. Fig. 9 presents a visualization that identifies dissonances between the problem and the solution space. The visualization displays inconsistencies between the current feature model and its implementation through different views: the dependency matrix shows inconsistencies between feature dependencies in red (top-right); the feature model shows suggested feature dependencies with colored lines (middle-right); the code-view displays the feature implementation (middle-left); and finally, the nodelink diagram depicts the code-level interactions (bottom).

3.2. Dimension 2: Visualization approach

Visualization can be regarded as the intervention to cope with increasingly complex evolution scenarios. By introducing Visual Analytics, decision makers can focus their attention on visualization-enabled analytical reasoning while benefiting from automatic data processing techniques. This dimension can be characterized along with the sort of interaction type [24] and the visualization strategy [10].

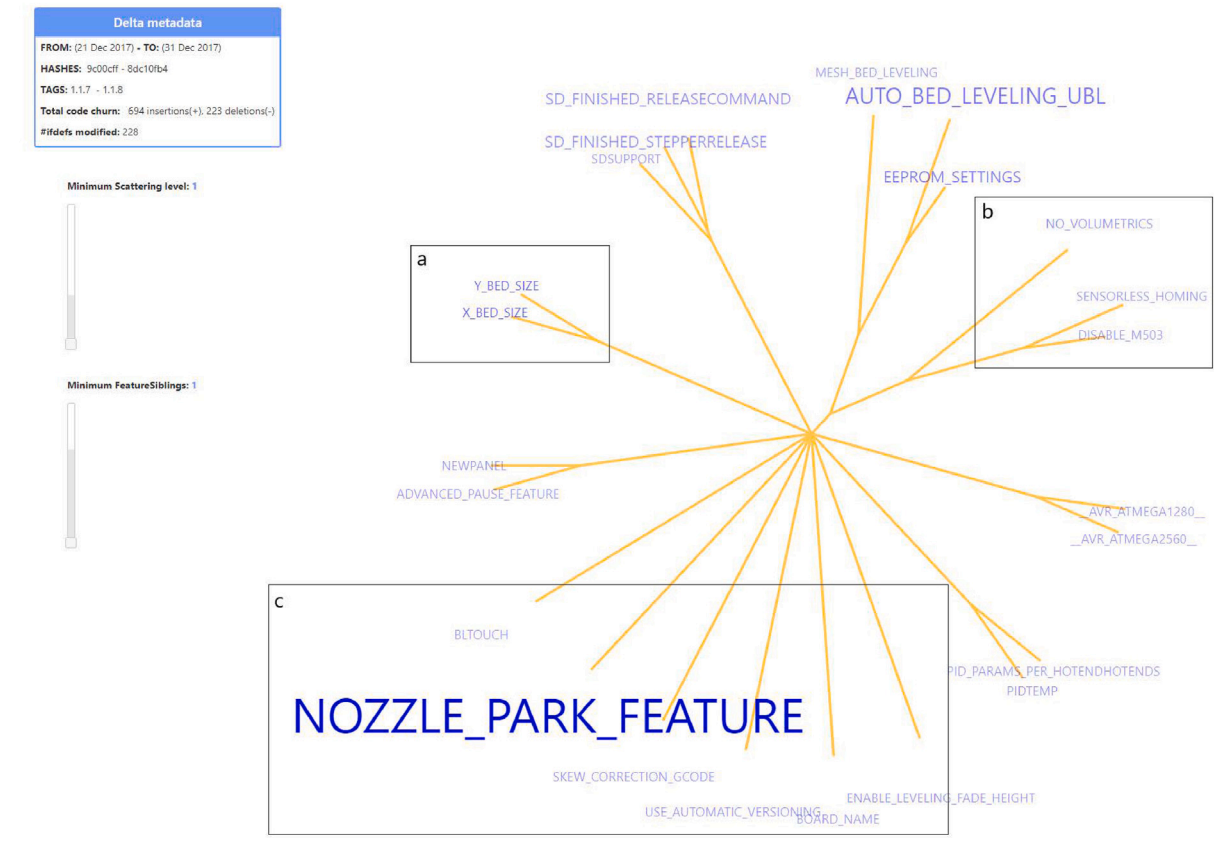


Fig. 7. *Functional Testing. FeatureCloud* helps identify hot-spot features, i.e., features undergoing substantial changes in their codebase from the last release of the SPL platform. *FeatureCloud* takes the code churn, and extracts the pre-compilation directives from the associated *ifdef* blocks. On these grounds, the tool generates a word-cloud using feature names as words, and pre-compilation directives as phrases. Nodes stand for features. Color opacity, font size, and distribution of nodes denote the size of the change, the scattering degree and the tangling degree, respectively, for the feature at hand. Take the case of feature *AUTO_BED_LEVELING_UBL* (right-upper corner in the screenshot). From the last release, this feature has been subject to few changes (light-blue color). Yet, testing is guessed to be complicated. First, this feature's changes are *scattered* along distinct *ifdef* blocks (as reflected by the medium font size). Second, these changes pertain to *ifdef* blocks whose pre-compilation directives involve feature *AUTO_BED_LEVELING_UBL* but also other five features that are depicted in the same radial branch, e.g., *EEPROM_SETTING*. Figure credits [S7]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

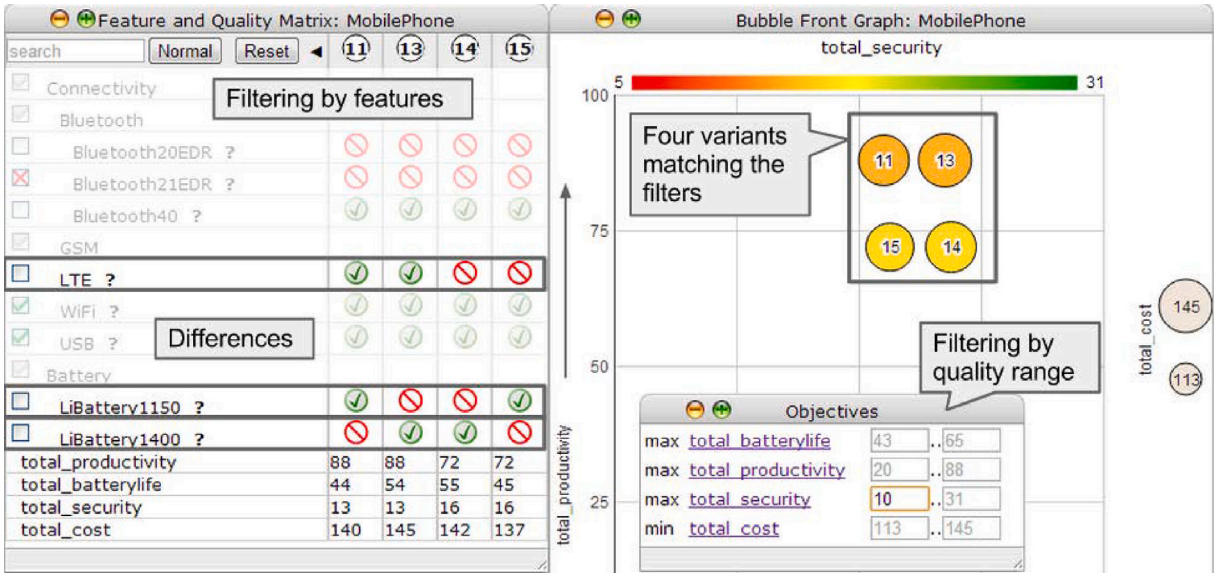


Fig. 8. *Analysis of Non-Functional Properties. Bubble Front Graphs* display four custom quality (non-functional) metrics in a bubble chart. This way, users can select the optimal variants based on those quality metrics. The chart uses four dimensions to represent those quality metrics: horizontal axis X (bottom), the vertical axis Y (left), bubble color Z (top) and bubble size *T* (right) to visualize up to four quality dimensions simultaneously. In the figure, productivity is represented as a vertical coordinate (Y); batterylife is represented as a horizontal coordinate (X); security is represented as a color (Z); and cost is represented as a size (T). Figure credits [S8]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

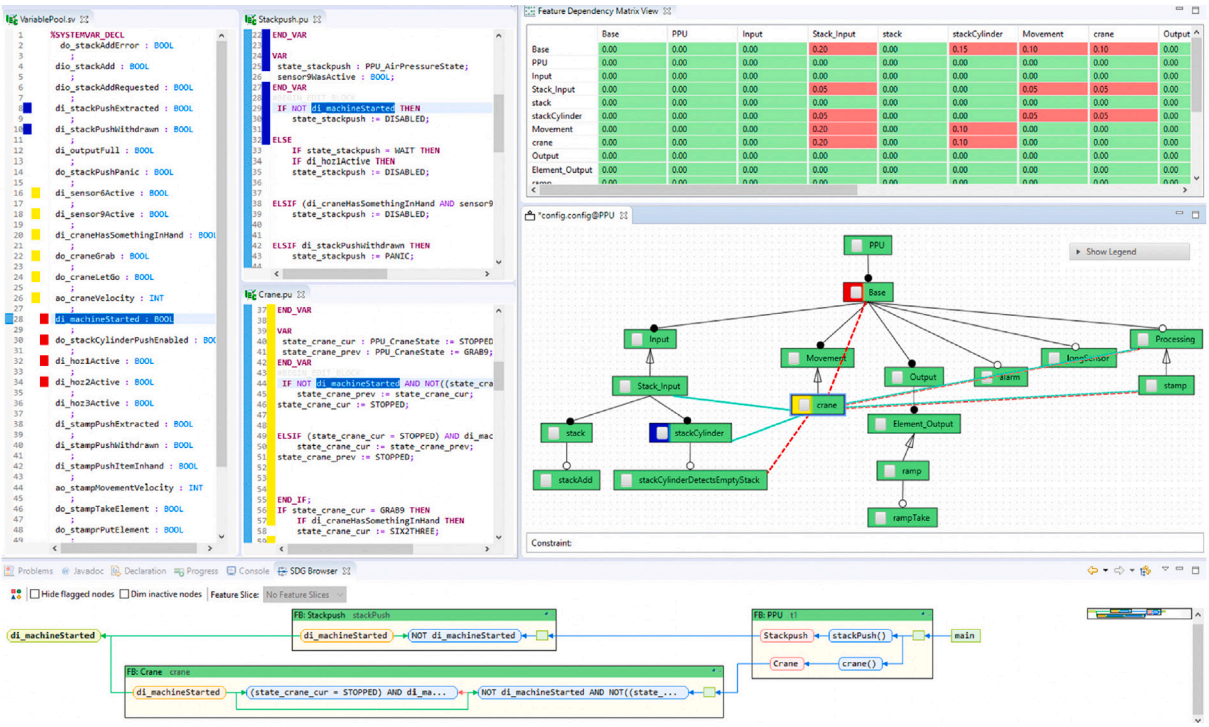


Fig. 9. *Co-evolution of Problem Space and Solution Space.* This visualization guides developers on the evolution of the feature model by showing them inconsistencies between the current feature model and its implementation. The top-right matrix displays the inconsistencies (marked in red). The visualization also suggests feature-level dependencies which are shown in the feature model on the middle-right (colored lines). Developers can also consult the implementation of the selected feature on the upper-left part and check its code-level dependencies in the nodelink diagram on the lower part. Figure credits [S9]. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Visualization interaction. Yi et al. introduce the following interaction schema [24]:

- *Filter*. When the visualization allows users to specify several conditions that the data that is shown must satisfy.
- *Select*. When the visualization provides users with the ability to focus on a specific data item(s).
- *Abstract/Elaborate*. When the visualization provides users with the functionality of adjusting the abstraction/detail level of the data.
- *Connect*. When the visualization provides users with the option of checking associations and relationships between data items.
- *Explore*. When the visualization enables users to explore a different subset of data items.
- *Reconfigure*. When the visualization allows users to watch different perspectives by rearranging the data.

Visualization strategy. Novais et al. introduce the following strategy schema [10]:

- *Temporal Snapshot (TS)*. It represents a snapshot of the software at a specific point in time.
- *Temporal Overview (TO)*. It displays all of the values of the development of a particular metric for all of the software entities being examined. This complicates the presentation since we must represent several entities (e.g. software modules) in a single visual picture, as well as the temporal evolution of one or more of their properties (e.g. size).
- *Temporal Accumulative Snapshot (TA)*. It considers the absolute value of changes to study the evolution of the program. It is commonly employed in the study of software churn [25].
- *Differential Relative (DR)*. It expresses the increase or decrease in a software module's property. If a software feature f1 increased its scattering by 2 points, but another feature f2 increased it by 4, then the DR representation should graphically show that f2 grew more than f1. In a similar way, decrements should be presented.
- *Differential Absolute (DA)*. When the definition of growth or reduction does not apply, the absolute strategy comes in handy. It is focused on recognizing and representing discrete events, such as entity properties that have appeared or vanished from one software version to the next.

3.3. Dimension 3: Research maturity

No matter the analysis nor the visualization, an important insight is the extent of the evidence of the intervention utility. To this end, we resort to two existing classifications for Research Type [26] and Evaluation Method [27]. In addition, we consider the Publication Venue as an additional proxy of the research maturity.

Research type. Wieringa et al. introduce the following research type schema [26]:

- *Evaluation research*. Explains a method of evaluating research that involves assessing how a solution works in practice or comparing it to other solutions while highlighting positive and negative aspects. It is more comprehensive than validation and is frequently performed in an industrial context.
- *Validation research*. Describes the validation of a research that has not yet been implemented in practice, for example, by an experiment, the performance of tests, lab experiments, and so on. It usually comes after a proposal for a solution.
- *Proposal of solutions*. Without a thorough validation, these works suggest a solution approach and argue for its utility. The approach must be new, or at the very least a considerable improvement on one that already exists. A proof-of-concept might be presented in the form of a simple example, a compelling argument, or another method.
- *Philosophical papers*. These publications propose a new way of thinking, a new conceptual framework, etc.

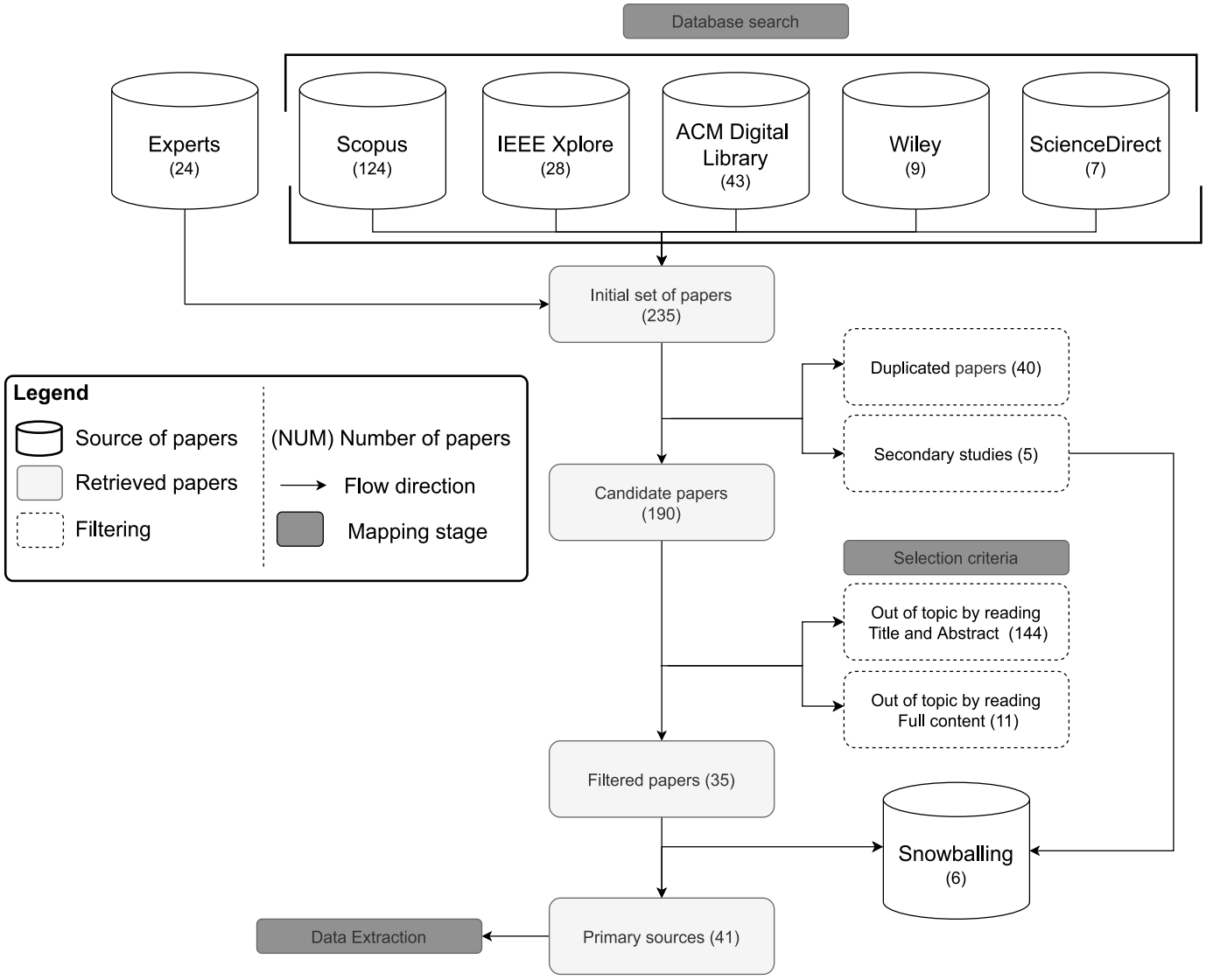


Fig. 10. Primary sources selection process.

- *Personal experience papers*. Represent the authors' experience, generally in practice, with a particular method, technology, or other device. These papers are frequently authored by industry professionals.

Evaluation method. Stol and Fitzgerald introduce the following schema [27]:

- *Field Studies*. Any research undertaken in a specific, real-world situation to explore a specific software engineering issue.
- *Field Experiments*. Experimental studies conducted in a natural setting with a high degree of realism (similar to a field study), but this evaluation method reduces the level of realism compared to a field study because the researcher manipulates some properties or variables in the research setting in order to observe an effect of some kind.
- *Experimental Simulations*. In this kind of evaluations, the behaviors of actors (e.g., developers, users, or software systems) that a researcher aims to observe and measure in experimental simulations are natural, but the setting in which they occur has been specifically created for the purposes of the study—that is, the setting would not exist if the study did not exist.
- *Laboratory Experiments*. Laboratory experiments differ from field experiments in that field experiments explore phenomena in their natural setting, whereas laboratory experiments are conducted in a controlled environment.
- *Judgment Studies*. These studies entail obtaining empirical data from a group of people who are asked to assess or rate actions, reply to a researcher's request or 'stimulus', or debate a particular topic of interest.
- *Sample Studies*. These are studies that try to attain generalizability across a certain demographic or set of factors, such as software professionals, software systems, or development process artifacts.
- *Formal theory*. This is a way of evaluation that aims for a high level of universality so that the resultant theory or framework may be used in a wide range of situations, even though most theories have boundaries beyond which they do not apply.
- *Computer Simulation*. Studies that replicate a real-world phenomenon or setting using a computer. These kinds of studies are like virtual laboratory experiments where hypotheses can be tested.

Publication venue. We distinguish between four different venue types: journals, conferences, workshops and symposiums.

4. Research design

The goal of this study is to review the visualization interventions that have been proposed to assist in the maintenance and evolution of VRSs. To this end, we follow the recommendations presented in [6]. This section describes the plan for each step of the study, including the research questions, academic databases and search strategy, and inclusion/exclusion criteria.

4.1. Research questions

RQ1: *What sort of analysis is being conducted to assess VRS evolution?*. RQ1 takes an analysis perspective. In accordance with the Goal-Question-Metric (GQM) model, an analysis is conducted by an agent with a purpose through a set of questions grounded on a collection of metrics [28]. Accordingly, we could characterize studies based on the agent ('target role'), the purpose ('evolution scenario'), and the support provided by the tool to conduct the analysis ('added-value'). We can then refine RQ1 as follows:

- **RQ1.1:** *Which are the target roles involved?*
- **RQ1.2:** *What are the evolution goals being pursued?*
- **RQ1.3:** *What is the added-value provided by the tool?*

RQ2: *What sort of visualization are displayed?*. RQ2 takes a tool perspective. The question intends to find out what kind of visualization tools are used on VRS evolution and what are the elements used to build them. This question can be refined as follows

- **RQ2.1:** *What are the elements being visualized?*
- **RQ2.2:** *Which visualization techniques are used?*
- **RQ2.3:** *Which sort of visual interactions are catered for?*
- **RQ2.4:** *What visualization strategies are used?*
- **RQ2.5:** *What data sources are used to build which visualizations?*
- **RQ2.6:** *To what extent is VRS scalability considered in each evolution scenario?*

RQ3: *What is the research maturity of the reported interventions?* No matter the analysis nor the visualization, this question sheds light on the degree of maturity of the primary sources.

4.2. Search strategy

Search string. Along the lines of Kitchenham et al. [6], we broke down the question into individual facets, namely, VRS as 'the population', evolution as 'the aim', and 'Visual Analytics' (i.e., visualization) as 'the intervention'. We searched for synonyms. In addition, we noticed that 'maintenance' and 'evolution' are terms that are arbitrarily interchanged. We also noticed that visualization is often expressed in other forms such as 'visual' or 'visualizing'. Hence, we included visual with a wildcard *. What was more elusive was the notion of VRS as it encompasses areas such as product lines, variability management and configurable systems. This resulted on the following search string:

("evol" OR "maint*") AND ("visual*") AND ("product line" OR "feature model" OR "variability management" OR "configurable system")*

Academic databases. Primary sources were obtained by querying the following databases: Scopus,[2] ACM Digital Library,[3] IEEEXplore,[4] ScienceDirect[5] and Wiley Online Library[6] (see the top of Fig. 10). We did not query Google Scholar and SpringerLink as they lacked the advanced query functionality we needed.

[2] https://www.scopus.com/.
[3] https://dl.acm.org/.
[4] https://ieeexplore.ieee.org/.
[5] https://www.sciencedirect.com/search.
[6] https://onlinelibrary.wiley.com/.

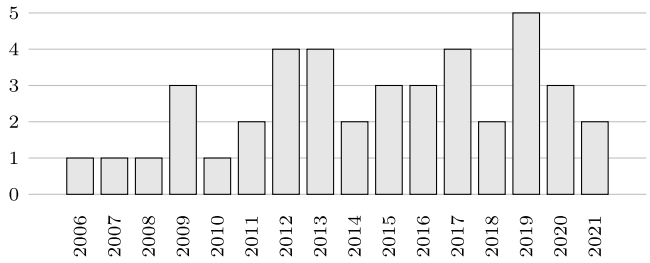


Fig. 11. Distribution of the primary sources per year.

Search scope. Search was conducted upon papers' title, abstract, and keywords. To check the validity of the search string, we manually double checked a few articles.

4.3. Protocol

Fig. 10 outlines the process. We ran the query on the 17th of September of 2021. We obtained an initial set of 234 papers from which 24 belonged to the papers manually selected by the authors. At the onset, we removed duplicated papers (40) and secondary studies. This resulted in 189 candidate papers ('Candidate papers'). We reserved secondary studies for a snowballing phase.

Candidate papers (189) went next through a filtering process. The title and the abstract were carefully read by two researchers (the first two authors). In case of doubt, the introduction and conclusion, or even the full content of the paper was also read. Each paper was filtered based on the inclusion and exclusion criteria described in Table 1. Snowballing was conducted for the remaining 35 papers that provided an addition of 6 papers. This gives a total of 41 primary sources. To minimize potential biases on the selection process, two other researchers (the last two authors) were involved when doubts or disagreements arouse between the other two researchers. This happened for 15 papers. Finally, two researchers (the two first authors) carried out the data extraction process. Each paper classification was surveyed by at least another researcher. Fig. 11 shows the distribution of the primary sources publication per year. It suggests a sustained effort throughout with an average of 2.75 publications per year.

5. Data extraction & Paper classification

Data from primary sources was extracted along the template in Table 2. The table indicates the Research Questions, the items extracted to answer them, the domain ('type of data') and cardinality ('multivalued') of these items and the section where the information item is described. Besides the literature-backed classification schemas introduced in Section 3, new ones were derived from the data itself (*ad-hoc* schemas). That is, if an existing scheme did not exist, thematic analysis was conducted by grounding classification themes on quotes extracted directly from the primary sources. Next, we review these *ad-hoc* schemas.

RQ1.1: Target role. Regarding 'target role', a main distinction in SPLs is between domain engineers and application engineers. As the review scope is not limited to SPLs but includes VRSs in general, we conducted a fine-grained categorization of 'target roles' through a coding step (i.e., grouping equivalents, synonyms, and assigning ids). We identified six categories, namely:

- *Application Engineers*: The people who develop variants out of existing code. Primary sources refer to them as either 'application engineer' or 'product developer'.
- *Domain Engineers*: The people who develop code to be reused. Primary sources refer to them as 'SPL architects', 'domain engineers' or 'product line maintainer'.

Table 1
Exclusion and inclusion criteria.

Type	Criteria
Inclusion criteria - Focus on topic	– **The study focuses on evolution or maintenance.** – **The study focuses on variant-rich systems.** – **The study proposes/uses some kind of graphical *visualization* technique to support evolution activities**
Inclusion criteria - Quality guarantee	– **The study is peer reviewed**
Exclusion criteria - Type of paper	– **The study is in a language other than English** – **The study is a previous version of another study in the review** – **The work is a Secondary study such as a literature review**
Exclusion criteria - Characteristics of the approach	– **The visualization is not tool supported.** We only focus on visualizations that have an implementation behind them, e.g., we exclude manually created graphs

Table 2
Data extraction template.

RQ	Extracted item	Type of data	Multi-valued	Described at
RQ1	Analysis scenarios	Yes	Yes	Section 3.1
	Target roles	Ad-hoc schema	Yes	Section 5
	Added-value	Ad-hoc schema	Yes	Section 5
RQ2	Visualized elements	Ad-hoc schema	Yes	Section 5
	Visualization technique	Ad-hoc schema	Yes	Section 5
	Visualization interaction	Existing schema	Yes	Section 3.2
	Visualization strategies	Existing schema	Yes	Section 3.2
	Data sources	Ad-hoc schema	Yes	Section 5
RQ3	Research type	Existing schema	No	Section 3.3
	Evaluation method	Existing schema	Yes	Section 3.3
	Tool availability	Free text	No	Section 5
Metadata	General meta-data	Free text	No	Section 5

- *Software Engineers*: This code is used when authors do not make a clear distinction between Application Engineers and Domain Engineers. This code abstracts from quotes referring to either 'engineers' or 'code developers'.
- *Requirement Engineers*: This code is used when the primary sources explicitly mention that the tool targets people in charge of requirement engineering activity. Primary sources refer to them as 'requirement engineer' or 'scope analyst'.
- *Testers*: Here, authors mention that the tool targets testers or it is used for testing purposes.
- *Project Managers*: The people in charge of the planning, scheduling, budgeting, execution, and delivery of a software. This code abstracts from quotes referring to either 'managers', 'change control board' or 'product line managers'.

RQ1.3: Added-value. This facet accounts for the degree of support provided by the visualization from merely displaying the raw data to engineering analysis workflows as part of the tool offerings. Codes follow:

- *Raw*. When raw elements are used in the visual analysis (e.g., codebase), at best, tool interactions are limited to provide some sort of data aggregation,
- *Metric*. When metrics are computed from the raw elements as proxies for the analysis (e.g., number of methods, feature cohesion),
- *Analysis*. When some sort of analysis is being conducted out of the metric (e.g., data-flow analysis, feature-location analysis).

RQ2.1: Visualized elements. Visualized elements may coincide with the data sources. Sometimes however, elements such as features are abstracted out of the code without the existence of a feature model.

RQ2.3: Visualization technique. Examples of visualization techniques include nodelink diagrams, heatmaps, bubble charts, etc.

RQ2.5: Data sources. This includes [10], namely: *'source code'* (e.g., the SPL platform codebase or the variant codebase, if available), *'feature model'*, *'variant configuration model'*, other model-based artifacts (e.g., UML notation such as architectural-component diagrams or state-transition diagrams), *'traceability links'* (feature-code traces, requirement-variant traces), *'project documentation'* (requirements or design documents), *'VCS meta-data'* (commits, issues), and the *'run-time context'* (logs).

RQ3.3. Tool availability/replicability. We checked the availability of the visualization tool (source code or installer package). First, we searched for an URL in the paper that would take us to the tool. If broken, we contacted the authors asking for an active link. If the link was not present, we googled it using the tool name as the search string.

General meta-data. We recovered the venues and publication years for the primary sources.

6. Results for RQ1: What sort of analysis is being conducted to assess VRS evolution?

An analysis is conducted by an agent ('target role') with a purpose ('evolution scenario') assisted by a visualization tool ('added-value'). We delve into each of these characteristics.

6.1. Target roles

Table 3 outlines the extent each target role is being referred to in the primary sources. *'Domain engineers'* and *'application engineers'* get the lion's share: 31 of the primary sources target these roles. For the rest of the primary sources, either this distinction did not apply (e.g., clone & own where a reusability platform is not available, and hence the 'target role' of *'domain engineer'* does not yet exist) or they tackle marginal cases (e.g., testing, management).

6.2. Evolution scenarios

Table 4 summarizes the results. We can observe how visualization support develops in tandem with the VRS field. At the onset, VRSs depart from clone & own to an extractive approach on the search for systematic reuse. Here, *'Feature Identification and Location'* in existing variants was (and still is) a main endeavour (10 papers). As the area matures, and VRS develops towards configuration systems and SPLs,

Table 3
Target roles numbers.

Target role	Number	Primary sources
Application engineers	16	[S10,S9,S11,S7,S2,S12,S13,S14,S5,S15,S3,S16,S17,S6,S18,S19]
Domain engineers	15	[S9,S7,S20,S4,S21,S22,S8,S23,S19,S24,S25,S26,S15,S25,S27]
Software engineers	9	[S28,S29,S30,S31,S32,S33,S34,S35,S1]
Testers	3	[S36,S37,S16]
Requirements engineers	2	[S38,S39]
Project managers	2	[S40,S41]

Table 4
Evolution scenarios numbers.

Evolution scenario	Number	Primary sources
Feature Identification and Location (FIL)	10	[S28,S31,S32,S35,S3,S22,S38,S17,S39,S13]
Variant Integration (VI)	8	[S26,S30,S2,S33,S21,S14,S41,S23]
Variant Synchronization (VS)	6	[S10,S27,S12,S11,S37,S1]
Functional Testing (FT)	6	[S30,S7,S36,S29,S16,S19]
Transformations (TR)	5	[S25,S11,S24,S20,S6]
Co-Evolution of Problem and Solution Spaces (CPS)	4	[S9,S7,S40,S18]
Constraints Extraction (CE)	4	[S9,S4,S34,S15]
Architecture Recovery (AR)	2	[S5,S19]
Analysis of Non-Functional Properties (ANF)	1	[S8]

'Variant Synchronization' *'Variant Integration'*, and *'Transformations'* are moved to the spot (19 papers). Visualization wise, efforts are made to understand and support evolution rather than on testing the preservation of certain (non)functional properties. Interestingly, it is not uncommon for primary sources to investigate multiple scenarios: [S30] (*VI* and *FT*), [S11] (*VS* and *TR*) or [S19] (*FT* and *AR*).

6.3. Added-value

Table 5 displays the results: 8 papers contribute through elaborated mining approaches to provide the raw data; 14 papers place a visualization on top of the raw data to highlight different metrics; finally, 18 papers contribute by providing some sort of analysis on top of the visualization. This distribution 8-14-18 is akin to the role played by each added-value: one sort of raw data might be aggregated along different metrics which, in turn, might allow for different analysis perspectives.

We complement this report by listing the metrics and analysis types at play: code churn [S23,S2,S7], feature scattering [S26,S7,S31], number of feature dependencies [S9,S18], code delta metric [S23], feature cohesion metric [S6], feature size [S26], feature tangling [S7], feature model metrics [S20], number of method overload [S35], runtime non-functional metrics [S8], significance criteria [S10], t-wise coverage [S36] and weighted hamming distance [S16]. On the other hand, analysis strategies include, formal concept analysis [S21,S38], n-way analysis [S5,S17,S3,S14], behavior analysis and comparison [S12], bug prevalence analysis [S37], clone detection analysis [S39], data-flow analysis [S34], feature location analysis [S32,S13], MOEA Analysis [S25], NLP analysis [S22], pair-wise feature relations [S4], PLC project analysis [S28], scope analysis [S41], and variant similarity analysis [S33,S1,S39].

With the exception of [S23,S39,S7,S26], papers focus on either a single metric or a single analysis strategy. This pattern is common in the early stages of an area subject to analysis (as it is the case of VRS evolution). Rationales are many-fold: mining techniques are not yet mature; lack of evidence about the appropriate metrics; and the scarcity of real scenarios where to validate the analysis strategies. As the area matures, we expect more hybrid approaches where different metrics and strategies are combined to tackle more realistic settings.

7. Results for RQ2: What sort of visualization are displayed?

7.1. Visualized elements (RQ2.1)

Table 6 displays the results: *'features'*, *'variants'*, and *'implementation assets'* (e.g., source code classes) are the most popular. These elements seem to satisfy the analysis needs of the most investigated scenarios (see Section 6.2), i.e., *'Variant Synchronization'* and *'Feature Identification and Location'*.

7.2. Visualization techniques (RQ2.2)

Table 7 summarizes the results: *'nodelink diagram'* (e.g. Figs. 6 and 4), and *'tabular/matrix'* (e.g., Fig. 8) are by far the most popular visualization techniques. This coincides with visualizations for one-off development [10]. In total, we found 11 different visualization techniques. We note that 16 primary sources implemented more than one: *'heatmap'* and *'enriched software diagram'* (see Fig. 9) [S9]; *'tabular/matrix'* and *'enriched software diagram'* [S26]; *'tabular/matrix'* and *'line chart'* [S11]; *'alluvial diagram'* and *'bar chart'* (see Fig. 2) [S2]; *'tabular/matrix'* and *'bar chart'* [S31,S14]; *'tabular/matrix'*, *'bubble chart'*, *'treemap'* and *'heatmap'* [S36]; *'tabular/matrix'* and *'bubble chart'* (see Fig. 8) [S8]; *'nodelink diagram'* and *'tabular/matrix'* [S34,S40]; *'bar chart'*, *'heatmap'* and *'treemap'* (see Fig. 3) [S3]; *'tabular/matrix'* and *'heatmap'* [S16]; *'bar chart'* and *'nodelink diagram'* (see Fig. 1) [S1]; *'nodelink diagram'* and *'word cloud'* (see Fig. 7) [S7]; *'tabular/matrix'*, *'bar chart'* and *'enriched software diagram'* (see Fig. 5) [S5]; or *'alluvial diagram'* and *'treemap'* [S23].

7.3. Visualization interactions (RQ2.3 & RQ2.6)

Table 8 outlines the results. Nearly half of the primary sources (44%) do not support any kind of interaction but just provide static diagrams. For those supporting some interaction, *'abstract/elaborate'* is the most common. This may be due to the different levels of abstraction present in VRSs (e.g. variants, features, source code). Some of the primary sources start the analysis at the variant level and next, drill-down to either the features or the source code [S30,S2,S1,S39,S23,S12,S5,S3]. Both *'filter'* and *'select'* mechanisms are also widely used, akin to the large codebase of VRSs. Finally, *'connect'* and *'explore'* are the least popular which could be due to the yet poorly elaborated analysis frameworks for managing VRS evolution.

7.4. Visualization strategies (RQ2.4)

Table 9 summarizes the results: the vast majority of primary sources use a *'temporal snapshot'* strategy. Tentatively, researchers are also considering other scenarios, e.g., *'temporal overview'* and *'differential relative'*. Exceptionally, three primary sources support various strategies: [S9,S11] both support *'temporal snapshot'* and *'differential relative'*, while [S1] supports *'temporal snapshot'* and *'temporal overview'*. The ability to change the strategy based on user-demand seems to be a desirable property for future visualization tools.

Table 5
Added-value numbers.

Added-value	Number	Primary sources
Analysis strategy	19	[S21,S38,S5,S17,S12,S37,S39,S34,S32,S13,S25,S22,S3,S14,S4,S41,S33,S1,S28]
Metrics	14	[S23,S2,S7,S26,S31,S9,S18,S6,S20,S35,S8,S10,S36,S16]
Raw	8	[S27,S30,S11,S24,S29,S40,S15,S19]

Table 6
Visualized elements numbers.

Visualized element	Number	Primary sources
Features	21	[S9,S30,S11,S7,S2,S31,S32,S24,S20,S4] [S36,S34,S15,S22,S17,S6,S39,S41,S19,S23,S13]
Variants	19	[S10,S27,S11,S2,S12,S33,S36,S8,S21,S37] [S14,S3,S22,S38,S16,S17,S1,S39,S23]
Implementation assets	15	[S9,S28,S30,S11,S31,S32,S12,S35,S3,S1] [S6,S39,S19,S23,S13]
Feature model	5	[S9,S26,S20,S40,S18]
Architecture	4	[S25,S24,S29,S5]
Variation points	4	[S27,S28,S15,S35]
Requirements	2	[S21,S38]

Table 7
Visualization techniques numbers.

Visualization technique	Number	Primary sources
Nodelink diagram	23	[S11,S32,S12,S13,S29,S34,S40,S15,S35,S6,S39] [S18,S19,S33,S21,S38,S4,S30,S1,S20,S25,S7,S24]
Tabular/Matrix	11	[S11,S31,S36,S8,S26,S34,S14,S40,S16,S39,S41]
Bar chart	7	[S10,S31,S3,S17,S1,S14,S2]
Enriched software diagram	5	[S9,S26,S37,S5,S25]
Heatmap	4	[S9,S36,S3,S16]
Treemap	4	[S36,S3,S23,S13]
Indented tree	3	[S27,S28,S24]
Word cloud	2	[S7,S22]
Bubble chart	2	[S36,S8]
Alluvial diagram	2	[S2,S23]
Line chart	1	[S11]

Table 8
Visualization interactions numbers.

Visualization interaction	Number	Primary sources
None	17	[S10,S9,S25,S26,S31,S32,S13,S24,S33,S36,S21,S37] [S34,S15,S22,S38,S41]
Abstract/Elaborate	14	[S28,S30,S11,S7,S2,S12,S20,S14,S5,S3,S16,S1,S6,S23]
Select	10	[S27,S28,S11,S4,S29,S16,S1,S6,S39,S19]
Filter	8	[S30,S7,S33,S4,S40,S35,S17,S18]
Explore	5	[S11,S7,S8,S15,S39]
Connect	2	[S28,S30]
Reconfigure	1	[S6]

Table 9
Visualization strategies numbers.

Visualization strategy	Number	Primary sources
Temporal snapshot	33	[S21,S38,S16,S17,S1,S6,S39,S18,S19,S14,S34,S40,S5,S15,S35,S3,S22] [S10,S9,S25,S27,S28,S30,S11,S31,S32,S12,S13,S24,S20,S4,S36,S8]
Temporal overview	5	[S33,S29,S37,S1,S41]
Differential relative	5	[S9,S26,S7,S2,S23]
Differential absolute	1	[S11]

7.5. Data sources (RQ2.5)

Table 10 shows the results: *'source code'* and *'feature model'* are the most common data sources with 29 and 17 primary sources, respectively. *'Traceability links'* and *'models'* are also relevant, but with half of the occurrences. In the case of *'traceability links'*, researchers resort to different approaches: *'feature-to-code'* [S26,S30,S11,S32,S6, S19], *'feature-to-component'* [S24,S19], *'feature-to-variant'* [S19,S13], or *'requirement-to-variant'* [S21,S19].

Researchers often resort to a combination of data sources. Fig. 12 depicts the most popular combinations using an Upset plot [29]. The combinations *('source code', 'feature model', 'traceability links')* is explored in 4 primary sources. We expect more research on how to combine different data sources.

7.6. Visualization techniques & Data sources (RQ2.5)

This section breaks down visualization techniques by data sources (see Fig. 13 (right-side)). Questions of interest include: What sources are the most used with *tabular/matrix*? What techniques do usually depend on *traceability links*? Fig. 13 (right side) depicts the outcome. We excluded visualization techniques which involve less than three data sources. It comes as no surprise that the *'source code'* and *'nodelink*

Table 10
Data sources numbers.

Data source	Number	Primary sources
Source code	29	[S35,S3,S1,S6,S39,S19,S23,S13,S15,S28,S16,S10,S21,S37,S22,S27] [S9,S26,S30,S11,S7,S2,S31,S32,S12,S33,S34,S14,S5]
Feature model	17	[S9,S27,S26,S30,S11,S20,S4,S36,S8,S15,S16,S6,S19,S32,S24,S13,S41]
Traceability links	9	[S26,S30,S11,S32,S6,S19,S24,S21,S13]
Models	8	[S25,S24,S29,S40,S15,S6,S19,S17]
VCS metadata	5	[S2,S37,S19,S23,S18]
Project documentation	4	[S21,S38,S25,S8]
Variant config. model	3	[S4,S29,S16]
Runtime contexts	2	[S30,S29]

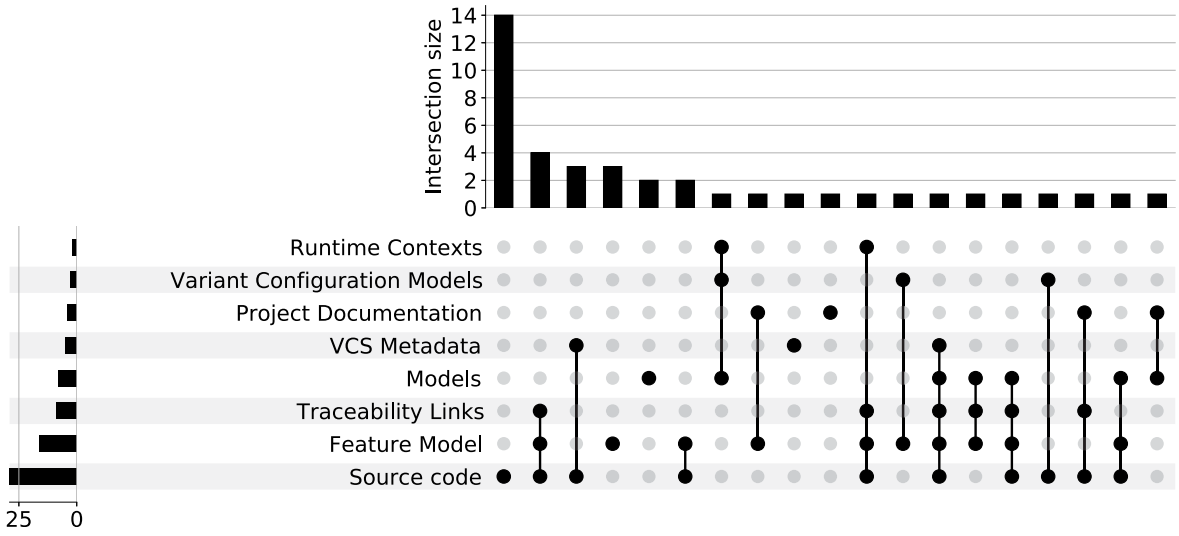


Fig. 12. Upset plot of data sources combinations. The bar chart on the left represents the number of primary sources using a particular data source, while the bar chart on the top represents the number of primary sources with a visualization resorting to a particular combination of one or more data sources.

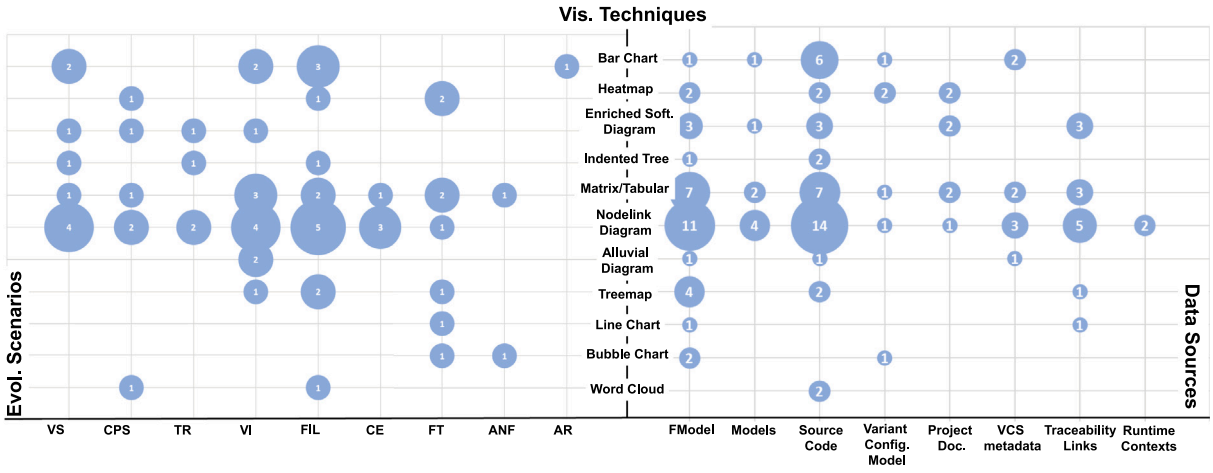


Fig. 13. Visualization techniques distribution by evolution scenarios and data sources.

diagram' have the biggest relation as they are the most popular approaches. *'Feature model'* for *'nodelink diagram'* follows the lead with 11 occurrences. In general, *'feature model'* and *'source code'* are the most popular data sources no matter the visualization technique. The exception is that of *'enriched software diagram'*. Here, *'feature model'*, *'source code'* and *'traceability links'* are equally popular.

7.7. Visualization techniques & Evolution scenario (RQ2.2)

This section breaks down visualization techniques by evolution scenario (see Fig. 13 (left-side)). *'Nodelink diagram'* and *'tabular/matrix'* are widely supported throughout the six scenarios. By contrast, *'alluvial diagram'* and *'line chart'* techniques are used in a single scenario, specifically, *'Variant Integration'* and *'Functional Testing'*, respectively.

7.8. Assessing scenario maturity (RQ2.6)

It could be argued that the more sophisticated the interaction, the more mature are the scenarios being targeted by the tool. On these premises, this section assesses scenario maturity based on the sort of supported interactions. Specifically, we grouped *Interaction* codes

Table 11
Research types numbers.

Research type	Number	Primary sources
Solution proposal	20	[S29,S11,S12,S13,S14,S3,S16,S17,S18,S19,S20,S23,S24,S25] [S27,S39,S34,S1,S36,S40]
Validation research	14	[S28,S10,S9,S7,S2,S6,S21,S22,S8,S26,S30,S32,S33,S35]
Evaluation research	5	[S5,S15,S4,S38,S41]
Conceptual proposal	1	[S37]
Experience paper	1	[S31]

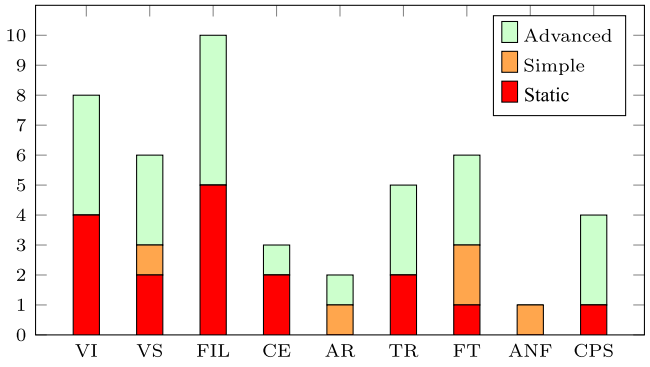


Fig. 14. Evolution scenarios vs. visualization interactions. *Advanced* interaction refers to 'Abstract/Elaborate', 'Reconfigure', and 'Filter', while *Simple* includes 'Explore', 'Connect', and 'Select'.

in two groups: advanced and simple. The former encompasses *'abstract/elaborate'*, *'reconfigure'* and *'filter'* as they tackle large visualization spaces. By contrast, simple interactions include *'explore'*, *'connect'* and *'select'* that handle smaller data-set, once they have been filtered. On these premises, we could define a 'maturity model' for visualization scenarios that move from 'static' (i.e., no interaction) to 'simple dynamic' and finally, 'advance dynamic'. Fig. 14 depicts this journey for each of the scenarios. *'Variant Integration'*, *'Variant Synchronization'*, *'Feature Identification and Location'* and *'Co-Evolution of Problem Space and Solution Space'* exhibit the larger rate of advanced interactions. Rationales might be the fact that these scenarios face large sets of elements to be displayed, and hence, the need for advanced interactions is self-evident.

8. Results for RQ3: What is the research maturity of the reported interventions?

This section considers four proxies to assess research maturity: (1) Research Type, (2) Evaluation Method, (3) availability of the tool, and (4) Publication Venue.

8.1. Research Type

Table 11 shows the distribution of the Research Types along with Wieringa et al.'s schema [26]. *'Solution proposal'* is the category with more primary sources (20). Works falling into this category limit their contribution to presenting the visualization tool through a proof of concept. The primary sources conducting some kind of evaluation move to either *'validation research'* (14) or *'evaluation research'* (5). These studies represent nearly half of the primary sources.

8.2. Evaluation Method

Table 12 delves into the sort of evaluation method: *'field experiment'* is the most popular method with 7 primary sources. Only four primary sources conducted a *'field study'* which is considered the strongest evaluation.

Table 12
Evaluation methods numbers.

Evaluation method	Number	Primary sources
Field experiment	7	[S28,S10,S9,S6,S4,S26,S38]
Judgment studies	4	[S2,S8,S30,S32]
Laboratory experiment	3	[S5,S21,S22]
Experimental simulation	3	[S7,S33,S35]
Field study	2	[S15,S41]

8.3. Tool availability

Making a tool available for public scrutiny and download is certainly a proof of confidence about the degree of completion of the tool. It also permits replicability and reproducibility, and in so doing, allows for knowledge accumulation. At the moment of conducting the mapping, 13 tools were available in the paper or through a quick search on the internet (see Table 13). We also include tools available upon request.

8.4. Publication Venue

Different sorts of venues account for distinct objectives. While workshops welcome proof-of-concept manuscripts at an early stage of research, journals tend to request sounder proof of utility and empirical evidence. Table 14 shows the distribution per venue. The majority of the primary studies (64%) appeared in conferences, SPLC being the most popular one. Only three found their way into journals. We conjecture that this situation is due to the difficulty of researchers in providing good-enough empirical evidence. Of the proposed tools, 53.7% lack a strong empirical evaluation in terms of the realism about either the VRS at hand, the analysis tasks being conducted, or the subjects being used in the experiments. Besides the complexity that goes in building the tool, researchers face the difficulty of the lack of available industrial VRSs.

9. Discussion

This section provides some insights about what we feel as promising research directions.

Project managers role. *'Project managers'* are in charge of making important decisions about the evolution of the VRS. It seems logical to think that those decisions should be tool supported. However, the results show that only two primary sources target this role. Visual Analytics plays a key role in decision making in other areas [30] and VRSs evolution should be no exception.

Overlooked scenarios. Table 4 highlights that scenarios *'Analysis of Non-Functional properties'* and *'Architecture Recovery'* have been scarcely studied: one and two primary studies, respectively. We think that more attention should be dedicated to these two scenarios. A loose analysis of non functional properties may impact on crucial aspects such as the security or the efficiency of the VRS [31]. As for *'Architecture Recovery'*, it is an essential step in the transit from clone & own to an SPL platform. Though reverse engineering techniques have been proposed for building a reference platform architecture [32], these techniques might benefit from Visual Analytics.

Table 13
Primary sources with available tools.

Primary source	Tool availability
[S4]	https://github.com/but4reuse/frogs, https://github.com/FeatureIDE since 3.3
[S5]	Initially available under request. Contact Fraunhofer IESE.
[S6]	http://www.fosd.de/FeatureVisu
[S7]	https://github.com/rmedeiros/FeatureCloud
[S8]	https://github.com/gsdlab/claferMooVisualizer
[S34]	http://www.cin.ufpe.br/~emergo
[S10]	https://issue-propagation.github.io/demo/
[S11]	https://bitbucket.org/easelab/featuredashboard
[S17]	Integrated in https://but4reuse.github.io/
[S20]	http://smalltalkhub.com/abergel/Familiar/
[S22]	Integrated in https://but4reuse.github.io/
[S24]	Integrated in https://github.com/xLineMapper
[S25]	https://github.com/otimizes/OPLA-Tool
[S28]	https://www.isf.cs.tu-bs.de/cms/team/schlie/material/VAToolkit
[S32]	Available on request from the authors

Table 14
Venues where the primary sources were published. Venues are ordered by type.

Venue	Number	Primary sources	Type
Software and Systems Modeling (SoSyM)	1	[S19]	Journal
IEICE Transactions on Information and Systems	1	[S33]	Journal
Journal of Computer Languages (COLA)	1	[S9]	Journal
Int. Systems and Software Product Line Conf. (SPLC)	9	[S11,S7,S2,S16,S22] [S8,S35,S1,S37]	Conference
Working Conf. on Software Visualization (VISSOFT)	4	[S13,S20,S4,S36]	Conference
Int. Conf. on Aspect-oriented software development (AOSD)	2	[S38,S34]	Conference
Conf. on Software Engineering and Advanced Applications (SEAA)	2	[S40,S39]	Conference
Working Conf. on Reverse Engineering (WCRE)	2	[S14,S5]	Conference
Int. Conf. on Req. Engineering (RE)	1	[S41]	Conference
Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)	1	[S29]	Conference
Int. Conf. on Software Reuse (ICSR)	1	[S15]	Conference
European Conf. on Modelling Foundations and Applications (ECMFA)	1	[S17]	Conference
Int. Conf. on Software Engineering (ICSE)	1	[S6]	Conference
Int. Working Conf. on Req. Engineering: Fdn. for Soft. Quality (REFSQ)	1	[S26]	Conference
Int. Conf. on Industrial Cyberphysical Systems (ICPS)	1	[S27]	Conference
Conference on Enterprise Information Systems (CENTERIS)	1	[S21]	Conference
Int. Workshop on Reverse Variability Engineering (REVE)	3	[S28,S23,S31]	Workshop
Int. Workshop on Variability Modelling of Software-intensive Sys. (VaMoS)	1	[S32]	Workshop
Int. Workshop on Req. Engineering Visualization (REV)	1	[S18]	Workshop
Int. Workshop on Knowledge-Oriented Product Line Engineering (KOPLE)	1	[S3]	Workshop
ACM Symposium on Applied Computing (SAC)	1	[S24]	Symposium
Brazilian Symposium on Software Engineering (SBES)	1	[S25]	Symposium
Symposium on Conceptual Modelling Education (SCME-iStar)	1	[S12]	Symposium
Pacific Visualization Symposium (PacificVis)	1	[S10]	Symposium
Seminar on Advanced Techniques Tools for Software Evolution (SATToSE)	1	[S30]	Symposium

Furthermore, Struber et al. also include another evolution scenario: Feature Model Synthesis (FMS) [23]. FMS is the automated process for generating a first candidate for a feature model. Curiously, none of the primary sources tries to support this scenario. This could be because FMS is usually performed through automatic techniques [23] and graphs such as the ones created through formal concept analysis are used so far only as the internals for this automation. However, FMS usually relies on displaying alternatives feature model structures. We consider that visualizations can help during this decision process.

Promising visualization techniques. All the primary sources provide 2D visualizations, where interactions, if available, are rather traditional (e.g., selecting, filtering, etc.). However, computer visualization is experimenting a boom of novel techniques [9]. This includes dynamic/live visualization [33], augmented reality [34] or virtual reality [35], to name a few. Often, these techniques are introduced to cope with an increasing complexity in both the structure and quantity of data. VRSs as complex systems might be a good target where these techniques can be put to the test.

Promising data sources.

- *User Feedback*. We miss data sources used in other Software Engineering practices such as Continuous Software Engineering (CSE). CSE is an agile software development process that supports the incremental implementation of requirements and their timely validation through user feedback [36]. For example, in CSE decision knowledge and usage knowledge are two important information sources that drive the evolution and have been used in visualizations for CSE [37]. Both, usage and decision knowledge are used by practitioners while evolving their systems [38]. Again, VRSs might be a promising venue to look into the potential of these data sources.
- *Version Control Systems (VCS)*. Mining VCSs is an already mature area with several mining tools available [39,40]. This fact makes even more curious the lack of primary sources resorting to VCS data (such as issue trackers or commit messages). This could be related to the lack of available and open source VRSs.

Scalability awareness.

- *VRSs' large codebase*. Static visualization might be the start but they could hardly scale up to the hundreds of artifacts that a VRS encompasses. More investigation is needed in advanced interaction mechanisms that permit analysts *'abstract/elaborate'*, *'reconfigure'* and *'filter'* to spot 'bad smells' among the myriad of artifacts. In general, only 56% of the primary sources implement interaction mechanisms. This is a low percentage, yet similar to the rate presented by Novais et al. in one-off development [10].
- *VRSs' co-existence between the Problem Space (Feature Model) and the Solution Space (Configuration model, codebase)*. One-off products lack the notion of domain or family. By contrast, VRSs tackle the notion of family as a set of variants united and systematically managed from a common platform: 'family' and 'variant' co-exists throughout. Both notions feedback each other during evolution. Visualization wise, most primary sources focus on one view, either the family perspective or the variant perspective. As VRSs mature from configurable systems to full-fledged SPLs, this double perspective would become paramount for domain engineering to be informed of what is going on in application engineering, and vice versa, how variant upgrades end up being merged back in the SPL platform.
- *VRSs' large feature set*. If VRSs exhibit hundreds of features, then scalability is certainly a concern. Yet, most popular scenarios such as *VI*, *VS* and *FIL* exhibit a worrying situation; more or less a half of the primary sources tackling them ignore the scalability issue.

Visualization intervention evaluation. The fact that most tools are only partially validated, if validated at all, shows a worrying situation. This scenario is shared with visualization interventions for one-off development. In their literature review about software visualization, Merino et al. [41] found that 62% of the primary sources lack a strong evaluation. In the same vein, our review indicates that only 21% of the visualizations are validated in realistic industrial settings. In this respect, research methodologies such as Action Design Research might help involve practitioners. [42] provides a case in point.

10. Threats to validity

This section discusses four types of threats of validity: internal validity, external validity, construct validity, and conclusion validity [43].

Internal validity. Obtaining a representative set of literature publications for this systematic mapping study can be viewed as a validity threat due to the search process. To minimize this threat, we followed the guidelines proposed in [6]. We might have overlooked relevant search terms, and the search engines against which we performed the search could lack relevant venues. To limit these threats, we also bootstrapped the search using manually gathered ('experts') articles, and we performed snowballing. Another threat is our filtering step that could have resulted in relevant articles being filtered out. To limit this threat we use additional reviewers in case of doubt and consensus among the reviewers. We also allowed ourselves to read the whole article in case of doubt during the filtering phase.

External validity. Most of the primary sources are only authored by academics. This might back findings being useful for academia, but the doubt remains about its impact in industry. Also, our findings are mainly within the field of VRS evolution. We cannot generalize our results beyond tasks other than evolution, and software other than VRS.

Construct validity. Threats related to the construct validity are the suitability of the research questions and the categorization scheme used to extract the data. To mitigate these threats, we resorted to external taxonomies as much as possible (such as the one proposed in Strüber et al. [23], Novais et al. [10], Wieringa et al. [26] and Stol and Fitzgerald [27]). As for the questions, the issues raised are similar to those tackled in other literature reviews [10] but accommodated to VRS specifics.

Conclusion validity. Concerning the subjectivity of the assessment of the primary sources, they were reviewed by at least two authors to mitigate bias in data extraction. In case of disagreements, the researchers discussed these cases to reach consensus. We also used multiple reviewers during the classification phase to limit the subjectivity of the process.

11. Conclusions

The inherent complexity in the evolution of variant-rich systems calls for dedicated visualizations. To this end, we analyzed 41 visualizations to answer three main questions about (1) the purpose of the analysis being conducted, (2) the kind of supporting visualization, and (3) the maturity of the reported interventions.

Analysis wise, we found that researchers tend to overlook *'project managers'* role and three evolution scenarios: *'Feature Model Synthesis'*, *'Analysis of Non-Functional Properties'*, and *'Architecture Recovery'*.

Visualization wise, we analyzed visual elements, visualization techniques, interactions, strategies and data sources. We found 11 different visualization techniques where the *'nodelink diagram'* was the most popular with 23 primary sources. Unfortunately, scalability seems to be an aspect with high potential for improvement as only 56% of the primary sources implement interaction mechanisms. When it comes to strategies and data sources, *'temporal snapshot'* and *'source code'* are the most used approaches, respectively. Overall, we miss studies applying visualization techniques with increasing popularity (e.g. 3D or Virtual Reality) and data sources that are used in other Software Engineering areas (e.g. user feedback or decision knowledge). Finally, benchmarks might also help provide common ground to assess and compare visualization interventions. All in all, Visual Analytics for variant-rich systems looks promising, yet in an early stage of development. We hope this work will have an impact in planning and categorizing future contributions to the field.

CRediT authorship contribution statement

Raul Medeiros: Conceptualization, Methodology, Validation, Resources, Writing – original draft, Investigation, Writing – review & editing, Visualization, Project administration. **Jabier Martinez:** Conceptualization, Methodology, Validation, Resources, Investigation, Writing – review & editing, Data curation. **Oscar Díaz:** Conceptualization, Methodology, Investigation, Writing – review & editing, Supervision. **Jean-Rémy Falleri:** Conceptualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by the Spanish Ministry of Science, Innovation and Universities grant number RTI2018-099818-B-I00. Raul Medeiros enjoys a doctoral grant from the Spanish Ministry of Science and Innovation (PRE2019-087324).

Appendix. Related literature surveys

See Table A.15.

Table A.15
Characterization of related literature surveys.

Literature survey	Target artifact	Research topic	Research questions	# Papers
Kienle and Müller [7]	One-off	Requirements of Vis. Tools for evolution	What (non-functional) requirements should software visualization tools satisfy?	Unmentioned
Shahin et al. [8]	One-off	Software architectures	What visualization techniques are used in software architecture? How much evidence is available to support the adoption of different visualization techniques in software architecture? What are the different purposes of using visualization techniques in software architecture? What are the domains in which architecture visualization techniques have been applied to support architecting activities? Which are the architecture visualization techniques mostly used in industry?	53
Chotisarn et al. [9]	One-off	Software processes	What software systems aspects are visually supported? What software engineering processes are visually supported? What are the software engineering roles involved? What visualization techniques have been used? What display mediums have been used? To what extent are visualization tools reusable?	105
Novais et al. [10]	One-off	Software evolution	What maintenance tasks are current SEV technologies evidently supporting and how do the approaches differ from each other?	146
Lopez-Herrejon et al. [13]	SPL	SPLs	In what stages of the SPL life cycle have visualization techniques been used? What visualization techniques have been used? What visualization tools have been used? What are the publication fora used?	32
Lopez-Herrejon et al. [11]	SPL	SPLs	The same as in [13] plus: What forms of empirical evaluation have been used? What is the provenance of evaluation examples used?	37
This work	VRS	VRS evolution	See Section 4.1	41

References

[1] T. Mens, S. Demeyer (Eds.), Software Evolution, Springer, 2008, http://dx.doi.org/10.1007/978-3-540-76440-3.

[2] W. Mahmood, D. Strüber, T. Berger, R. Lämmel, M. Mukelabai, Seamless variability management with the virtual platform, in: 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021, IEEE, 2021, pp. 1658–1670, http://dx.doi.org/10.1109/ICSE43902.2021.00147.

[3] T. Berger, J. Steghöfer, T. Ziadi, J. Robin, J. Martinez, The state of adoption and the challenges of systematic variability management in industry, Empir. Softw. Eng. 25 (3) (2020) 1755–1797, http://dx.doi.org/10.1007/s10664-019-09787-6.

[4] K.A. Cook, J.J. Thomas, Illuminating the Path: The Research and Development Agenda for Visual Analytics, Technical Report, Pacific Northwest National Lab.(PNNL), Richland, WA (United States), 2005.

[5] S. Reddivari, S. Rad, T. Bhowmik, N. Cain, N. Niu, Visual requirements analytics: a framework and case study, Requir. Eng. 19 (3) (2014) 257–279, http://dx.doi.org/10.1007/s00766-013-0194-3.

[6] B.A. Kitchenham, D. Budgen, P. Brereton, Evidence-Based Software Engineering and Systematic Reviews, Vol. 4, CRC Press, 2015.

[7] H.M. Kienle, H.A. Müller, Requirements of software visualization tools: A literature survey, in: J.I. Maletic, A.C. Telea, A. Marcus (Eds.), Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2007, Banff, Alberta, Canada, June 25-26, 2007, IEEE Computer Society, 2007, pp. 2–9, http://dx.doi.org/10.1109/VISSOF.2007.4290693.

[8] M. Shahin, P. Liang, M.A. Babar, A systematic review of software architecture visualization techniques, J. Syst. Softw. 94 (2014) 161–185, http://dx.doi.org/10.1016/j.jss.2014.03.071.

[9] N. Chotisarn, L. Merino, X. Zheng, S. Lonapalawong, T. Zhang, M. Xu, W. Chen, A systematic literature review of modern software visualization, J. Vis. 23 (4) (2020) 539–558, http://dx.doi.org/10.1007/s12650-020-00647-w.

[10] R.L. Novais, A. Torres, T.S. Mendes, M.G. Mendonça, N. Zazworka, Software evolution visualization: A systematic mapping study, Inf. Softw. Technol. 55 (11) (2013) 1860–1883, http://dx.doi.org/10.1016/j.infsof.2013.05.008.

[11] R.E. Lopez-Herrejon, S. Illescas, A. Egyed, A systematic mapping study of information visualization for software product line engineering, J. Softw. Evol. Process. 30 (2) (2018) http://dx.doi.org/10.1002/smr.1912.

[12] M. Raatikainen, J. Tiihonen, T. Männistö, Software product lines and variability modeling: A tertiary study, J. Syst. Softw. 149 (2019) 485–510, http://dx.doi.org/10.1016/j.jss.2018.12.027.

[13] R.E. Lopez-Herrejon, S. Illescas, A. Egyed, Visualization for software product lines: A systematic mapping study, in: B. Sharif, C. Parnin, J. Fabry (Eds.), 2016 IEEE Working Conference on Software Visualization, VISSOFT 2016, Raleigh, NC, USA, October 3-4, 2016, IEEE Computer Society, 2016, pp. 26–35, http://dx.doi.org/10.1109/VISSOFT.2016.11.

[14] M.A. Babar, L. Chen, F. Shull, Managing variability in software product lines, IEEE Softw. 27 (3) (2010) 89–91, http://dx.doi.org/10.1109/MS.2010.77.

[15] R. Mahdavi-Hezaveh, J. Dremann, L.A. Williams, Software development with feature toggles: practices used by practitioners, Empir. Softw. Eng. 26 (1) (2021) 1, http://dx.doi.org/10.1007/s10664-020-09901-z.

[16] S. Apel, C. Kästner, C. Lengauer, FEATUREHOUSE: Language-independent, automated software composition, in: 31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings, IEEE, 2009, pp. 221–231, http://dx.doi.org/10.1109/ICSE.2009.5070523.

[17] K. Czarnecki, U.W. Eisenecker, Generative Programming - Methods, Tools and Applications, Addison-Wesley, 2000, URL: http://www.addison-wesley.de/main/main.asp?page=englisch/bookdetails&productid=99258.

[18] D.S. Batory, J.N. Sarvela, A. Rauschmayer, Scaling step-wise refinement, IEEE Trans. Softw. Eng. 30 (6) (2004) 355–371, http://dx.doi.org/10.1109/TSE.2004.23.

[19] C. Prehofer, Feature-oriented programming: A new way of object composition, Concurr. Comput. Pract. Exp. 13 (6) (2001) 465–501, http://dx.doi.org/10.1002/cpe.583.

[20] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J. Loingtier, J. Irwin, Aspect-oriented programming, in: M. Aksit, S. Matsuoka (Eds.), ECOOP'97 - Object-Oriented Programming, 11th European Conference, Jyväskylä, Finland, June 9-13, 1997, Proceedings, in: Lecture Notes in Computer Science, vol. 1241, Springer, 1997, pp. 220–242, http://dx.doi.org/10.1007/BFb0053381.

[21] I. Schaefer, L. Bettini, V. Bono, F. Damiani, N. Tanzarella, Delta-oriented programming of software product lines, in: J. Bosch, J. Lee (Eds.), Software Product Lines: Going beyond - 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Proceedings, in: Lecture Notes in Computer Science, vol. 6287, Springer, 2010, pp. 77–91, http://dx.doi.org/10.1007/978-3-642-15579-6_6.

[22] R. Capilla, J. Bosch, P. Trinidad, A.R. Cortés, M. Hinchey, An overview of dynamic software product line architectures and techniques: Observations from research and industry, J. Syst. Softw. 91 (2014) 3–23, http://dx.doi.org/10.1016/j.jss.2013.12.038.

[23] D. Strüber, M. Mukelabai, J. Krüger, S. Fischer, L. Linsbauer, J. Martinez, T. Berger, Facing the truth: benchmarking the techniques for the evolution of variant-rich systems, in: T. Berger, P. Collet, L. Duchien, T. Fogdal, P. Heymans, T. Kehrer, J. Martinez, R. Mazo, L. Montalvillo, C. Salinesi, X. Tërnava, T. Thüm, T. Ziadi (Eds.), Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume a, Paris, France, September 9-13, 2019, ACM, 2019, pp. 26:1–26:12, http://dx.doi.org/10.1145/3336294.3336302.

[24] J.S. Yi, Y. ah Kang, J.T. Stasko, J.A. Jacko, Toward a deeper understanding of the role of interaction in information visualization, IEEE Trans. Vis. Comput. Graph. 13 (6) (2007) 1224–1231, http://dx.doi.org/10.1109/TVCG.2007.70515.

[25] T.L. Graves, A.F. Karr, J.S. Marron, H.P. Siy, Predicting fault incidence using software change history, IEEE Trans. Softw. Eng. 26 (7) (2000) 653–661, http://dx.doi.org/10.1109/32.859533.

[26] R.J. Wieringa, N.A.M. Maiden, N.R. Mead, C. Rolland, Requirements engineering paper classification and evaluation criteria: a proposal and a discussion, Requir. Eng. 11 (1) (2006) 102–107, http://dx.doi.org/10.1007/s00766-005-0021-6.

[27] K. Stol, B. Fitzgerald, The ABC of software engineering research, ACM Trans. Softw. Eng. Methodol. 27 (3) (2018) 11:1–11:51, http://dx.doi.org/10.1145/3241743.

[28] V.R. Basili, G. Caldiera, H.D. Rombach, The goal question metric approach, in: Encyclopedia of Software Engineering, Vol. 2, 1994, pp. 528–532, URL: http://www.csri.utoronto.ca/~sme/CSC444F/handouts/GQM-paper.pdf.

[29] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, H. Pfister, UpSet: Visualization of intersecting sets, IEEE Trans. Vis. Comput. Graphics (InfoVis) 20 (12) (2014) 1983–1992, http://dx.doi.org/10.1109/TVCG.2014.2346248.

[30] D.A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, H. Ziegler, Visual analytics: Scope and challenges, in: S.J. Simoff, M.H. Böhlen, A. Mazeika (Eds.), Visual Data Mining - Theory, Techniques and Tools for Visual Analytics, in: Lecture Notes in Computer Science, vol. 4404, Springer, 2008, pp. 76–90, http://dx.doi.org/10.1007/978-3-540-71080-6_6.

[31] M. Mukelabai, D. Nesic, S. Maro, T. Berger, J. Steghöfer, Tackling combinatorial explosion: a study of industrial needs and practices for analyzing highly configurable systems, in: M. Huchard, C. Kästner, G. Fraser (Eds.), Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018, ACM, 2018, pp. 155–166, http://dx.doi.org/10.1145/3238147.3238201.

[32] Z.T. Sinkala, M. Blom, S. Herold, A mapping study of software architecture recovery for software product lines, in: J. Pérez, R. Mirandola, H. Chen (Eds.), Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA 2018, Madrid, Spain, September 24-28, 2018, ACM, 2018, pp. 49:1–49:7, http://dx.doi.org/10.1145/3241403.3241454.

[33] A. Krause, M. Hansen, W. Hasselbring, Live visualization of dynamic software cities with heat map overlays, in: Working Conference on Software Visualization, VISSOFT 2021, Luxembourg, September 27-28, 2021, IEEE, 2021, pp. 125–129, http://dx.doi.org/10.1109/VISSOFT52517.2021.00024.

[34] A. Schreiber, L. Nafeie, A. Baranowski, P. Seipel, M. Misiak, Visualization of software architectures in virtual reality and augmented reality, in: 2019 IEEE Aerospace Conference, 2019, pp. 1–12, http://dx.doi.org/10.1109/AERO.2019.8742198.

[35] S. Romano, N. Capece, U. Erra, G. Scanniello, M. Lanza, On the use of virtual reality in software visualization: The case of the city metaphor, Inf. Softw. Technol. 114 (2019) 92–106, http://dx.doi.org/10.1016/j.infsof.2019.06.007.

[36] B. Fitzgerald, K. Stol, Continuous software engineering: A roadmap and agenda, J. Syst. Softw. 123 (2017) 176–189, http://dx.doi.org/10.1016/j.jss.2015.06.063.

[37] A. Kleebaum, B. Paech, J.O. Johanssen, B. Bruegge, Continuous rationale visualization, in: Working Conference on Software Visualization, VISSOFT 2021, Luxembourg, September 27-28, 2021, IEEE, 2021, pp. 33–43, http://dx.doi.org/10.1109/VISSOFT52517.2021.00013.

[38] J.O. Johanssen, A. Kleebaum, B. Paech, B. Bruegge, Continuous software engineering and its support by usage and decision knowledge: An interview study with practitioners, J. Softw. Evol. Process. 31 (5) (2019) http://dx.doi.org/10.1002/smr.2169.

[39] N.M. Tiwari, G. Upadhyaya, H.A. Nguyen, H. Rajan, Candoia: a platform for building and sharing mining software repositories tools as apps, in: J.M. González-Barahona, A. Hindle, L. Tan (Eds.), Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017, IEEE Computer Society, 2017, pp. 53–63, http://dx.doi.org/10.1109/MSR.2017.56.

[40] Y. Ma, T. Dey, C. Bogart, S. Amreen, M. Valiev, A. Tutko, D. Kennard, R. Zaretzki, A. Mockus, World of code: enabling a research workflow for mining and analyzing the universe of open source VCS data, Empir. Softw. Eng. 26 (2) (2021) 22, http://dx.doi.org/10.1007/s10664-020-09905-9.

[41] L. Merino, M. Ghafari, C. Anslow, O. Nierstrasz, A systematic literature review of software visualization evaluation, J. Syst. Softw. 144 (2018) 165–180, http://dx.doi.org/10.1016/j.jss.2018.06.027.

[42] O. Díaz, L. Montalvillo, R. Medeiros, M. Azanza, T. Fogdal, Visualizing the customization endeavor in product-based-evolving software product lines: a case of action design research, Empir. Softw. Eng. 27 (3) (2022) 75, http://dx.doi.org/10.1007/s10664-021-10101-6.

[43] X. Zhou, Y. Jin, H. Zhang, S. Li, X. Huang, A map of threats to validity of systematic literature reviews in software engineering, in: 2016 23rd Asia-Pacific Software Engineering Conference, APSEC, 2016, pp. 153–160, http://dx.doi.org/10.1109/APSEC.2016.031.

References (Primary sources)

[S1] V.L. Tenev, S. Duszynski, M. Becker, Variant analysis: Set-based similarity visualization for cloned software systems, in: M.H. ter Beek, W. Cazzola, O. Díaz, M.L. Rosa, R.E. Lopez-Herrejon, T. Thüm, J. Troya, A.R. Cortés, D. Benavides (Eds.), Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017, ACM, 2017, pp. 22–27, http://dx.doi.org/10.1145/3109729.3109753.

[S2] L. Montalvillo, O. Díaz, T. Fogdal, Reducing coordination overhead in spls: peering in on peers, in: T. Berger, P. Borba, G. Botterweck, T. Männistö, D. Benavides, S. Nadi, T. Kehrer, R. Rabiser, C. Elsner, M. Mukelabai (Eds.), Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018, ACM, 2018, pp. 110–120, http://dx.doi.org/10.1145/3233027.3233041.

[S3] J. Martinez, A.K. Thurimella, Collaboration and source code driven bottom-up product line engineering, in: E.S. de Almeida, C. Schwanninger, D. Benavides (Eds.), 16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2-7, 2012, Vol. 2, ACM, 2012, pp. 196–200, http://dx.doi.org/10.1145/2364412.2364445.

[S4] J. Martinez, T. Ziadi, R. Mazo, T.F. Bissyandé, J. Klein, Y.L. Traon, Feature relations graphs: A visualisation paradigm for feature constraints in software product lines, in: H.A. Sahraoui, A. Zaidman, B. Sharif (Eds.), Second IEEE Working Conference on Software Visualization, VISSOFT 2014, Victoria, BC, Canada, September 29-30, 2014, IEEE Computer Society, 2014, pp. 50–59, http://dx.doi.org/10.1109/VISSOFT.2014.18.

[S5] S. Duszynski, J. Knodel, M. Naab, D. Hein, C. Schitter, Variant comparison - a technique for visualizing software variants, in: A.E. Hassan, A. Zaidman, M.D. Penta (Eds.), WCRE 2008, Proceedings of the 15th Working Conference on Reverse Engineering, Antwerp, Belgium, October 15-18, 2008, IEEE Computer Society, 2008, pp. 229–233, http://dx.doi.org/10.1109/WCRE.2008.22.

[S6] S. Apel, D. Beyer, Feature cohesion in software product lines: an exploratory study, in: R.N. Taylor, H.C. Gall, N. Medvidovic (Eds.), Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu, HI, USA, May 21-28, 2011, ACM, 2011, pp. 421–430, http://dx.doi.org/10.1145/1985793.1985851.

[S7] O. Díaz, R. Medeiros, L. Montalvillo, Change analysis of #if-def blocks with featurecloud, in: C. Cetina, O. Díaz, L. Duchien, M. Huchard, R. Rabiser, C. Salinesi, C. Seidl, X. Tërnava, L. Teixeira, T. Thüm, T. Ziadi (Eds.), Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume B, Paris, France, September 9-13, 2019, ACM, 2019, pp. 59:1–59:4, http://dx.doi.org/10.1145/3307630.3342386.

[S8] A. Murashkin, M. Antkiewicz, D. Rayside, K. Czarnecki, Visualization and exploration of optimal variants in product line engineering, in: T. Kishi, S. Jarzabek, S. Gnesi (Eds.), 17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26-30, 2013, ACM, 2013, pp. 111–115, http://dx.doi.org/10.1145/2491627.2491647.

[S9] K. Feichtinger, D. Hinterreiter, L. Linsbauer, H. Prähofer, P. Grünbacher, Guiding feature model evolution by lifting code-level dependencies, J. Comput. Lang. 63 (2021) 101034, http://dx.doi.org/10.1016/j.cola.2021.101034.

[S10] Y. Kim, H. Jeon, Y. Kim, Y. Ki, H. Song, J. Seo, Visualization support for multi-criteria decision making in software issue propagation, in: 14th IEEE Pacific Visualization Symposium, PacificVis 2021, Tianjin, China, April 19-21, 2021, IEEE, 2021, pp. 81–85, http://dx.doi.org/10.1109/PacificVis52677.2021.00018.

[S11] S. Entekhabi, A. Solback, J. Steghöfer, T. Berger, Visualization of feature locations with the tool featuredashboard, in: C. Cetina, O. Díaz, L. Duchien, M. Huchard, R. Rabiser, C. Salinesi, C. Seidl, X. Tërnava, L. Teixeira, T. Thüm, T. Ziadi (Eds.), Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume B, Paris, France, September 9-13, 2019, ACM, 2019, pp. 55:1–55:4, http://dx.doi.org/10.1145/3307630.3342392.

[S12] A. Zamansky, I. Reinhartz-Berger, Visualizing code variabilities for supporting reuse decisions, in: X. Franch, M. Snoeck, R.S.S. Guizzardi, I. Jureta (Eds.), Proceedings of the 5th Symposium on Conceptual Modeling Education and the 2nd International iStar Teaching Workshop Co-Located with the 36th International Conference on Conceptual Modeling (ER 2017), Valencia, Spain, November 6-9, 2017, in: CEUR Workshop Proceedings, vol. 1954, CEUR-WS.org, 2017, pp. 25–34, URL http://ceur-ws.org/Vol-1954/SCME_2017_paper_5.pdf.

[S13] S. Illescas, R.E. Lopez-Herrejon, A. Egyed, Towards visualization of feature interactions in software product lines, in: B. Sharif, C. Parnin, J. Fabry (Eds.), 2016 IEEE Working Conference on Software Visualization, VISSOFT 2016, Raleigh, NC, USA, October 3-4, 2016, IEEE Computer Society, 2016, pp. 46–50, http://dx.doi.org/10.1109/VISSOFT.2016.16.

[S14] S. Duszynski, J. Knodel, M. Becker, Analyzing the source code of multiple software variants for reuse potential, in: M. Pinzger, D. Poshyvanyk, J. Buckley (Eds.), 18th Working Conference on Reverse Engineering, WCRE 2011, Limerick, Ireland, October 17-20, 2011, IEEE Computer Society, 2011, pp. 303–307, http://dx.doi.org/10.1109/WCRE.2011.44.

[S15] M. Sinnema, S. Deelstra, P. Hoekstra, The COVAMOF derivation process, in: M. Morisio (Ed.), Reuse of Off-the-Shelf Components, 9th International Conference on Software Reuse, ICSR 2006, Turin, Italy, June 12-15, 2006 Proceedings, in: Lecture Notes in Computer Science, vol. 4039, Springer, 2006, pp. 101–114, http://dx.doi.org/10.1007/11763864_8.

[S16] M. Al-Hajjaji, M. Schulze, U. Ryssel, Similarity analysis of product-line variants, in: T. Berger, P. Borba, G. Botterweck, T. Männistö, D. Benavides, S. Nadi, T. Kehrer, R. Rabiser, C. Elsner, M. Mukelabai (Eds.), Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018, ACM, 2018, pp. 226–235, http://dx.doi.org/10.1145/3233027.3233044.

[S17] J. Martinez, T. Ziadi, J. Klein, Y.L. Traon, Identifying and visualising commonality and variability in model variants, in: J. Cabot, J. Rubin (Eds.), Modelling Foundations and Applications - 10th European Conference, ECMFA@STAF 2014, York, UK, July 21-25, 2014. Proceedings, in: Lecture Notes in Computer Science, vol. 8569, Springer, 2014, pp. 117–131, http://dx.doi.org/10.1007/978-3-319-09195-2_8.

[S18] D. Sellier, M. Mannion, Visualizing product line requirement selection decisions, in: Software Product Lines, 11th International Conference, SPLC 2007, Kyoto, Japan, September 10-14, 2007, Proceedings. Second Volume (Workshops), Kindai Kagaku Sha Co. Ltd., Tokyo, Japan, 2007, pp. 109–118.

[S19] N. Anquetil, U. Kulesza, R. Mitschke, A. Moreira, J. Royer, A. Rummler, A. Sousa, A model-driven traceability framework for software product lines, Softw. Syst. Model. 9 (4) (2010) 427–451, http://dx.doi.org/10.1007/s10270-009-0120-9.

[S20] S. Urli, A. Bergel, M. Blay-Fornarino, P. Collet, S. Mosser, A visual support for decomposing complex feature models, in: 3rd IEEE Working Conference on Software Visualization, VISSOFT 2015, Bremen, Germany, September 27-28, 2015, IEEE Computer Society, 2015, pp. 76–85, http://dx.doi.org/10.1109/VISSOFT.2015.7332417.

[S21] T. Huysegoms, M. Snoeck, G. Dedene, A. Goderis, F. Stumpe, Visualizing variability management in requirements engineering through formal concept analysis, Proc. Technol. 9 (2013) 189–199, http://dx.doi.org/10.1016/j.protcy.2013.12.021, cENTERIS 2013 - Conference on ENTERprise Information Systems / ProjMAN 2013 - International Conference on Project MANagement/ HCIST 2013 - International Conference on Health and Social Care Information Systems and Technologies.

[S22] J. Martinez, T. Ziadi, T.F. Bissyandé, J. Klein, Y.L. Traon, Name suggestions during feature identification: the variclouds approach, in: H. Mei (Ed.), Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016, Beijing, China, September 16-23, 2016, ACM, 2016, pp. 119–123, http://dx.doi.org/10.1145/2934466.2934480.

[S23] L. Montalvillo, O. Díaz, M. Azanza, Visualizing product customization efforts for spotting SPL reuse opportunities, in: M.H. ter Beek, W. Cazzola, O. Díaz, M.L. Rosa, R.E. Lopez-Herrejon, T. Thüm, J. Troya, A.R. Cortés, D. Benavides (Eds.), Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017, ACM, 2017, pp. 73–80, http://dx.doi.org/10.1145/3109729.3109737.

[S24] G. Gharibi, Y. Zheng, Archfeature: integrating features into product line architecture, in: S. Ossowski (Ed.), Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, April 4-8, 2016, ACM, 2016, pp. 1302–1308, http://dx.doi.org/10.1145/2851613.2851764.

[S25] W.M. Freire, M. Massago, A.C. Zavadski, A.M.M.M. Amaral, T.E. Colanzi, Opla-tool v2.0: a tool for product line architecture design optimization, in: E. Cavalcante, F. Dantas, T. Batista (Eds.), SBES '20: 34th Brazilian Symposium on Software Engineering, Natal, Brazil, October 19-23, 2020, ACM, 2020, pp. 818–823, http://dx.doi.org/10.1145/3422392.3422498.

[S26] D. Hinterreiter, P. Grünbacher, H. Prähofer, Visualizing feature-level evolution in product lines: A research preview, in: N.H. Madhavji, L. Pasquale, A. Ferrari, S. Gnesi (Eds.), Requirements Engineering: Foundation for Software Quality - 26th International Working Conference, REFSQ 2020, Pisa, Italy, March 24-27, 2020, Proceedings [REFSQ 2020 was Postponed], in: Lecture Notes in Computer Science, vol. 12045, Springer, 2020, pp. 300–306, http://dx.doi.org/10.1007/978-3-030-44429-7_21.

[S27] J. Fischer, B. Vogel-Heuser, E. Estévez-Estévez, M. Male, Varapp: Variant management app for IEC 61131-3 compliant legacy software, in: IEEE Conference on Industrial Cyberphysical Systems, ICPS 2020, Tampere, Finland, June 10-12, 2020, IEEE, 2020, pp. 269–276, http://dx.doi.org/10.1109/ICPS48405.2020.9274774.

[S28] A. Schlie, K. Rosiak, O. Urbaniak, I. Schaefer, B. Vogel-Heuser, Analyzing variability in automation software with the variability analysis toolkit, in: C. Cetina, O. Díaz, L. Duchien, M. Huchard, R. Rabiser, C. Salinesi, C. Seidl, X. Tërnava, L. Teixeira, T. Thüm, T. Ziadi (Eds.), Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume B, Paris, France, September 9-13, 2019, ACM, 2019, pp. 89:1–89:8, http://dx.doi.org/10.1145/3307630.3342408.

[S29] D.H. Carmo, S.T. Carvalho, L.G.P. Murta, O. Loques, Runtime monitoring and auditing of self-adaptive systems (S), in: The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, June 27-29, 2013, Knowledge Systems Institute Graduate School, 2013, pp. 731–736.

[S30] B. Duhoux, K. Mens, B. Dumas, H.S. Leung, A context and feature visualisation tool for a feature-based context-oriented programming language, in: A. Etien (Ed.), Proceedings of the Seminar Series on Advanced Techniques & Tools for Software Evolution (SATTOSE 2019), Bolzano, Italy, July 8-10 Day, 2019, in: CEUR Workshop Proceedings, vol. 2510, CEUR-WS.org, 2019, URL http://ceur-ws.org/Vol-2510/sattose2019_paper_7.pdf.

[S31] J. Krüger, L. Nell, W. Fenske, G. Saake, T. Leich, Finding lost features in cloned systems, in: M.H. ter Beek, W. Cazzola, O. Díaz, M.L. Rosa, R.E. Lopez-Herrejon, T. Thüm, J. Troya, A.R. Cortés, D. Benavides (Eds.), Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B, Sevilla, Spain, September 25-29, 2017, ACM, 2017, pp. 65–72, http://dx.doi.org/10.1145/3109729.3109736.

[S32] B. Andam, A. Burger, T. Berger, M.R.V. Chaudron, Florida: Feature location dashboard for extracting and visualizing feature traces, in: M.H. ter Beek, N. Siegmund, I. Schaefer (Eds.), Proceedings of the Eleventh International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS 2017, Eindhoven, NetherlandS, February 1-3, 2017, ACM, 2017, pp. 100–107, http://dx.doi.org/10.1145/3023956.3023967.

[S33] T. Kanda, T. Ishio, K. Inoue, Approximating the evolution history of software from source code, IEICE Trans. Inf. Syst. 98-D (6) (2015) 1185–1193, http://dx.doi.org/10.1587/transinf.2014EDP7286.

[S34] M. Ribeiro, T. Tolêdo, J. Winther, C. Brabrand, P. Borba, Emergo: a tool for improving maintainability of preprocessor-based product lines, in: R. Hirschfeld, E. Tanter, K.J. Sullivan, R.P. Gabriel (Eds.), Companion Volume of the 11th International Conference on Aspect-Oriented Software Development, AOSD 2012, Potsdam, Germany, March 25-30, 2012, ACM, 2012, pp. 23–26, http://dx.doi.org/10.1145/2162110.2162128.

[S35] X. Tërnava, J. Mortara, P. Collet, Identifying and visualizing variability in object-oriented variability-rich systems, in: T. Berger, P. Collet, L. Duchien, T. Fogdal, P. Heymans, T. Kehrer, J. Martinez, R. Mazo, L. Montalvillo, C. Salinesi, X. Tërnava, T. Thüm, T. Ziadi (Eds.), Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume a, Paris, France, September 9-13, 2019, ACM, 2019, pp. 32:1–32:13, http://dx.doi.org/10.1145/3336294.3336311.

[S36] R.E. Lopez-Herrejon, A. Egyed, Towards interactive visualization support for pairwise testing software product lines, in: A.C. Telea, A. Kerren, A. Marcus (Eds.), 2013 First IEEE Working Conference on Software Visualization (VISSOFT), Eindhoven, the Netherlands, September 27-28, 2013, IEEE Computer Society, 2013, pp. 1–4, http://dx.doi.org/10.1109/VISSOFT.2013.6650543.

[S37] T.H.B. de Oliveira, M. Becker, E.Y. Nakagawa, Supporting the analysis of bug prevalence in software product lines with product genealogy, in: E.S. de Almeida, C. Schwanninger, D. Benavides (Eds.), 16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2-7, 2012 Volume 1, ACM, 2012, pp. 181–185, http://dx.doi.org/10.1145/2362536.2362561.

[S38] N. Niu, S.M. Easterbrook, Concept analysis for product line requirements, in: K.J. Sullivan, A. Moreira, C. Schwanninger, J. Gray (Eds.), Proceedings of the 8th International Conference on Aspect-Oriented Software Development, AOSD 2009, Charlottesville, Virginia, USA, March 2-6, 2009, ACM, 2009, pp. 137–148, http://dx.doi.org/10.1145/1509239.1509259.

[S39] T.F.L. de Medeiros, E.S. de Almeida, S.R. de Lemos Meira, Codescoping: A source code based tool to software product lines scoping, in: V. Cortellessa, H. Muccini, O. Demirörs (Eds.), 38th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2012, Cesme, Izmir, Turkey, September 5-8, 2012, IEEE Computer Society, 2012, pp. 101–104, http://dx.doi.org/10.1109/SEAA.2012.70.

[S40] R. Rabiser, D. Dhungana, W. Heider, P. Grünbacher, Flexibility and end-user support in model-based product line tools, in: 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009, Patras, Greece, August 27-29, 2009, Proceedings, IEEE Computer Society, 2009, pp. 508–511, http://dx.doi.org/10.1109/SEAA.2009.13.

[S41] K. Wnuk, B. Regnell, L. Karlsson, What happened to our features? visualization and understanding of scope change dynamics in a large-scale industrial setting, in: RE 2009, 17th IEEE International Requirements Engineering Conference, Atlanta, Georgia, USA, August 31 - September 4, 2009, IEEE Computer Society, 2009, pp. 89–98, http://dx.doi.org/10.1109/RE.2009.32.