A tertiary systematic literature review on Software Visualization

Laure Bedu, Olivier Tinh, Fabio Petrillo
Université du Québec à Chicoutimi
Chicoutimi, Canada
laure.bedu1@uqac.ca,olivier.tinh1@uqac.ca,fabio@petrillo.com

***Abstract*—Software visualization (SV) allows us to visualize different aspects and artifacts related to software, thus helping engineers understanding its underlying design and functionalities in a more efficient and faster way. In this paper, we conducted a tertiary systematic literature review to identify, classify, and evaluate the current state of the art on software visualization from 48 software visualization secondary studies, following three perspectives: publication trends, software visualization topics and techniques, and issues related to research field. Hence, we summarized the main findings among popular sub-fields of SV, identifying potential research directions and fifteen shared recommendations for developers, instructors and researchers. Our main findings are the lack of rigorous evaluation or theories support to assess SV tools effectiveness, the disconnection between tool design and their scope, and the dispersal of the research community.**

***Index Terms*—Software Visualization, Tertiary, Literature Review, SLR, Visualization techniques, Information Visualization, Recommendations, State of the art, Software Engineering**

I. Introduction

Software visualization (SV) is an Information Visualization domain related to software systems to understand visual representation of software architecture, behavior and evolution [49]. Besides algorithms and programs, artifacts related to software and its development problems are also visualized. Examples of such artifacts include requirements, bug reports, design documentation and so on. With such a broad definition, SV is associated with a wide range of software engineering domains.

Even though the SV has been studied from decades, many aspects are still uncertain or unexplored. It makes complex for researchers and practitioners to have a clear landscape of SV. The main goal of this paper is to identify the state of the art for understanding what we know about scientific research on SV. Researchers could gain from gathering and analyzing those different aspects of SV and consider them as a whole.

For achieving this goal, we applied the tertiary systematic literature review (SLR) methodology to provide a study that analyzes all secondary research contributions in the SV area. A tertiary literature review is a systematic process to extract and compile knowledge from secondary studies. A secondary study is a research work extract and compiles knowledge from primary studies. Primary studies are original materials on which results come from (controlled) experiments, surveys with practitioners, interviewers, mining repositories, static or dynamic analysis, and so ones.

In our study, we identified and evaluated the current state of the art on SV from different perspectives. We selected 48 secondary studies from over two hundred potentially relevant papers. Next, we rigorously analyzed the studies to extract our results. Finally, we summarized the results to produce a clear overview of the state of the art. Also, we identified potential research directions. We aimed at combining those different aspects to develop findings among it.

We consider secondary papers as papers based on primary sources (experiments and empirical studies) about a domain. Tertiary papers synthesize or summarize research that appears in secondary papers. Thus, our approach is to go beyond the concept of reviewing primary studies, reviewing secondary papers that already took a step back in SV concepts to produce reliable results. The study design is specified to allow replicability of this work. The purpose is to suggest even more reliable results, related to SV as a whole, identifying gaps in the field, for researchers to base on for future works. Important sub-field in which SV is applied are taken into consideration.

The main contributions of this study are: (i) a collection of 48 secondary studies on SV; (ii) an up-to-date map of state-of-the-art in SV and its implications for future research; (iii) a set of recommendations acting as sound guidelines for future development in SV field.

II. Background

SV is a sub-field of information visualization, which is the *use of computer-supported interactive visual representations of abstract data to amplify cognition* [50]. SV includes algorithm and program visualization (AV, PV), among others. The first one allows the visualization of general algorithms at a high level of abstraction while the latter focuses on the visualization of programs, usually at a lower level of abstraction [4].

There are numerous ways of representing information about software based on the purpose of the visualization. However, some techniques remained predominant such as graph-based representations as well as hierarchical ones. Besides, artifacts and architecture are mostly represented in SV systems, data sources coming mainly from Version Control Systems (VCS) and source codes [30], [34].

Beyond classical representation techniques, metaphors are also applied, which allow abstract concept representations through known entities. Those entities can consist of geometrical shapes or real-life objects such as cities, adopting practitioners well-known metaphors.

Visual design is a recurrent dilemma in SV tools conception. Choosing what kind of interaction could handle this to improve practitioners comprehension is a daunting challenge. Several papers refer to Schneiderman's mantra *"Overview first, zoom and filter, then show details on demand"* as a visualization strategy that should be supported [52].

III. Study design

Software Visualization is a vast field, so we follow recognized guidelines [51] to organize our SLR. Indeed, SLR have been proven efficient to make a software engineering state-of-the-art by using a systematic and reproducible process [55]. We explain our study design in the following sections.

A. Research Questions

Through this work, we answer those following three research questions:
RQ1 *What are the current publication characteristics in the software visualization research field?* Number of publications over the year, contribution types, main topics of selected papers are features that are characterized to answer this question.
RQ2 - *What are the application domain of software visualization?*
RQ2.1 - *For what purpose is software visualization used by developers, testers, project managers and even customers?* This question aims at identifying the reasons why software is visualized, what are the concerned fields on SV, and the software engineering related problems it can help resolve. In the corresponding section, we also mention the involved audience.
RQ2.2 *What/how is visualized using software visualization techniques?* This second part aims at defining the visualized metrics and the reasons why they are chosen among other choices. How they are visualized is related to commonly used techniques and metaphors among the previously identified purposes. Discussion regarding this question try to provide further research guidance, based on recurrent issues pointed out in our paper set.
RQ3 *What are the main issues regarding software visualization application?* Answering this question aims at identifying difficulties in terms of SV, limiting their use in specific domains, and tracks on how to overcome them in further researches.

B. Search and Selection of Secondary Studies

As a tertiary SLR, the search and selection process of the secondary studies are designed in seven steps, to control the number and attributes of selected studies. Figure 1 summarize those seven steps.
Raw search. First of all, we performed database searches, as recommended in SLR guideline papers [51], [53], on four indexing systems and scientific database related to software engineering: ACM Digital Library, IEEE Xplore, Scopus and Web of Science. We justify the use of these databases by their ability to export results in spreadsheet format, besides being considered as standard libraries [54]. The search strings

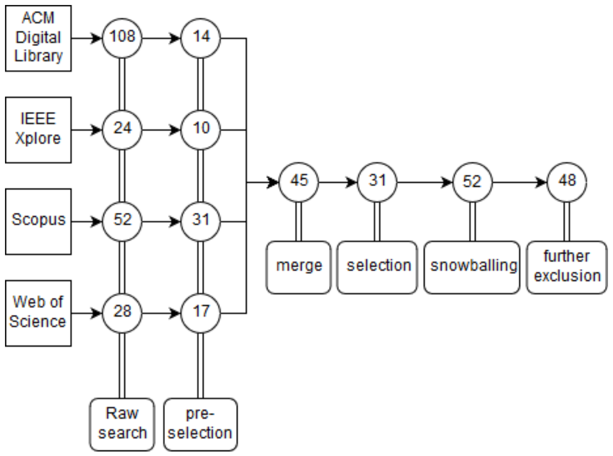

Fig. 1. Features on the search and selection process

we used are shown in Listing 1-4. Each one is divided into two parts, (i) one contains the most generic keyword that describes our subject (SV), (ii) the others is related to secondary study terms to exclude primary studies that are irrelevant to our tertiary approach. We wanted to control the number and relevance of the results. Therefore, the strings are limited to title, abstract and keywords.

Listing 1 – ACM DL
```
recordAbstract:(+"Software
visualization" "systematic review",
"literature review","systematic mapping"
,"mapping study", "systematic map",
"meta-analysis","survey",
"literature analysis")
```

Listing 2 – IEEE Xplore
```
("Software visualization" AND
("systematic review" OR
"literature review" OR
"systematic mapping" OR "mapping study"
OR "systematic map" OR "meta-analysis"
OR "literature analysis" OR "survey"))
```

Listing 3 – Scopus
```
TITLE-ABS-KEY ("Software visualization")
AND TITLE-ABS-KEY ("systematic review"
OR "literature review"
OR "systematic mapping"
OR "mapping study"
OR "systematic map" OR "meta-analysis"
OR "survey" OR "literature analysis")
AND (LIMIT-TO(SUBJAREA, "COMP"))
```

Listing 4 – Web of Science
```
ALL = ("Software visualization" AND
("systematic review"  OR
"literature review"  OR
"systematic mapping"  OR
"mapping study"  OR  "systematic map"
OR  "meta-analysis"  OR  "survey"
OR  "literature analysis"))
```

Pre-selection. We only considered journal and conference articles. For each result we obtained with the previous step, we studied titles and abstracts to find out whether they were related to SV. Irrelevant papers were thus excluded, establishing the first selection process, aiming at a coherent set of research studies.

Merging. In this phase, we combined the four database search results into a single data set. We matched raw data by title, authors and year before removing duplicated entries.

Selection. The selection consists of the application of a selection criteria. This step aims at making our procedure rigorous and reproducible, as advised in Kuhrmann et al. paper [54]. The data set is filtered according to the following criteria: I1 to I3 are inclusion criteria while E1 to E5 are exclusion ones.

I1 - Title, keyword list, and abstract explicitly stating that the paper is related to SV.
I2 - Studies conducting a systematic mapping study or SLR on SV or any sub-field.
I3 - Studies providing a state of the art, taxonomy, review on SV, or any sub-field.
E1 - Conference proceedings, e-books, slideshows, or formats different from research papers.
E2 - Papers focusing on a unique tool, technique, not broad enough.
E3 - The paper's full text is not available for download.
E4 - The paper is not in English.
E5 - The publication year is lower than 2000.

We accepted also SV co-relate terms and sub-fields as software evolution visualization, program visualization, or SV tools and techniques. We chose to include papers from 2000 to 2018 because they provide an overview of the evolution of the field throughout the years. The application of those criteria allows a time-effective and objective selection of the studies used in our literature review. Despite the criterion E5, we decided include Price *et al.* paper [3] because it is a major reference in SV, and it offers an interesting overview of the past state of the art of SV. Two authors applied the inclusion/exclusion criteria carefully, filtering papers that do not follow the criteria.

Snowballing. We conducted snowballing process in both direction (backward and forward) [51]. The main goal of this stage is to expand the set of possibly relevant papers by focusing on papers either citing or being cited by each previous selected studies. Backward snowballing proved to be iterative, as we first conduct this process right after the selection process on one depth level (we did not investigate the additional papers references), before adding relevant cited papers during the data extraction, thus increasing the depth level.

Further exclusions. Further exclusions concern papers that do not follow the inclusion/exclusion criteria while two-authors-reading the whole text during the data extraction process, because sometimes the previous reading of abstract and title did not reveal that the paper was out of the scope of our study.

C. Data Extraction

In this step, we gathered data from secondary papers through a well-defined process. Extracted data is organized according to the research question it answers.

RQ1. Publication trends: For each study we have recorded the publication year, the research strategy, and the main topics addressed. The definition of topics follows this process: for each paper, two authors read the title, abstract and more if necessary, and then define keywords that best describe the application domain. In case of a disagreement, a discussion occurs in to resolve conflicts.

RQ2.1 Software visualization goals: We had to define categories to represent the main sub-field that are represented in the studies set. They mainly come from previous main topics, if the number of papers related to a specific topic was not high enough, we did not consider them as categories, because the aim here is to combine results from papers dealing with the same subject. It does not claim to be exhaustive, although it suits well to the selected paper scope. Answering this question is fastidious, as SV tools and techniques are not necessary designed for a specific goal. Even well-known taxonomies as those presented by Roman et al. [59], and Price et al. [3], did not focus on this aspect of SV. Nevertheless, for each category we identified, we discuss main recurrent ideas that emerge from the selected studies SV goals and audience . We extracted other additional relevant information to develop the discussion.

RQ2.2 Metrics, techniques, metaphors: We employed the previous framework to discuss common techniques and metaphors employed in each previously defined domains. We also discuss emergent and innovative techniques, such as those exploiting 3D animation or virtual reality, as two systematic studies have been published on these subjects [17], [27].

RQ3. Software visualization application issues. Through full-text reading of our set of selected studies, we noted down issues related by authors. We took into account each issue emerging in at least two studies, and discuss it in the corresponding Section IV-C.

Finally, all selected papers are provided in Table I.

D. Data Synthesis

This activity mainly resulted in a synthesis of ideas encountered through metrics, techniques and task classification. It induces the building of our understanding of the results found.

E. Replicability of the Study

Through a meticulous description of the methodology employed in this work, researchers can replicate our process and widen the current limited number of studies reviewed (see Section VII). Section IV-B presents the framework used to classify results in thematic.

TABLE I: List of selected secondary studies

	Article	Topic	Year	Type
[3]	A principled taxonomy of software visualization	General	1993	Taxonomy
[35]	Software visualization tools: Survey and analysis	General	2001	Survey
[13]	A Task Oriented View of Software Visualization	General	2002	Classification
[20]	Documenting Software Systems with Views III: Towards a Task-oriented Classification of Program Visualization Techniques	General	2002	Classification
[32]	Software visualization	General	2005	Literature review
[26]	On the use of visualization to support awareness of human activities in software development: a survey and a framework	General	2005	Taxonomy
[37]	The paradox of software visualization	General	2005	Opinion paper
[48]	Visualization Techniques for Program Comprehension A Literature Review	General	2006	Literature review
[29]	Requirements of software visualization tools: A literature survey	General	2007	Literature review
[17]	An Overview of 3D Software Visualization	General	2009	Literature review
[25]	Mental imagery and software visualization in high-performance software development teams	General	2009	Survey
[46]	Visualization of the Static Aspects of Software: A Survey	General	2011	Literature review
[16]	An Information Visualization Feature Model for Supporting the Selection of Software Visualizations	General	2014	Literature review
[27]	Past, present, and future of 3D software visualization: A systematic literature analysis	General	2015	SLR[2]
[39]	To enlighten hidden facts in the code: A review of software visualization metaphors	General	2015	SLR
[34]	Software Visualization Today: Systematic Literature Review	General	2016	SLR
[40]	Towards Actionable Visualisation in Software Development	General	2016	SMS[1]
[43]	Visual augmentation of source code editors: A systematic mapping study	General	2018	Taxonomy
[22]	Exploring the Role of Visualization and Engagement in Computer Science Education	Education	2002	Taxonomy
[4]	A Review of Generic Program Visualization Systems for Introductory Programming Education	Education	2013	Literature review
[18]	Are Visualization Tools Used in Programming Education?: By Whom, How, Why, and Why Not?	Education	2014	Survey
[24]	Learning principles in program visualizations: A systematic literature review	Education	2016	SLR
[36]	Survey of software visualization systems to teach message-passing concurrency in secondary school	Education	2017	Literature review
[38]	Theoretical underpinnings of learner engagement in software visualization system: A systematic literature review protocol	Education	2018	SLR
[9]	A systematic literature review of student engagement in software visualization: a theoretical perspective	Education	2019	SLR
[14]	A Taxonomy of Computer Architecture Visualizations	Architecture	2002	Taxonomy
[1]	A Framework for Software Architecture Visualisation Assessment	Architecture	2005	Classification
[7]	A Survey Paper on Software Architecture Visualization	Architecture	2008	Literature review
[44]	Visualization and Evolution of Software Architectures	Architecture	2012	Literature review
[11]	A systematic review of software architecture visualization techniques	Architecture	2014	SLR
[5]	A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems	Evaluation	2009	Literature review
[21]	Evaluation of Software Visualization Tools	Evaluation	2009	Literature review
[41]	Validation of Software Visualization Tools: A Systematic Mapping Study	Evaluation	2014	SMS
[42]	Validation of the City Metaphor in Software Visualization	Evaluation	2015	Survey + Experimental
[8]	A systematic literature review of software visualization evaluation	Evaluation	2018	SLR
[33]	Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey	Maintenance	2003	Survey
[19]	Classifying Desirable Features of Software Visualization Tools for Corrective Maintenance	Maintenance	2008	Literature review
[6]	A survey on goal-oriented visualization of clone data	Maintenance	2015	Literature review
[28]	Program comprehension through reverse-engineered sequence diagrams: A systematic review	Maintenance	2018	SLR
[47]	Visualization Techniques for Application in Interactive Product Configuration	Product Line	2011	Literature review
[45]	Visualization for Software Product Lines: A Systematic Mapping Study	Product Line	2016	SMS
[10]	A systematic mapping study of information visualization for software product line engineering	Product line	2017	SMS
[2]	A meta-study of algorithm visualization effectiveness	Algorithm	2002	SMS
[15]	Algorithm Visualization: The State of the Field	Algorithm	2010	Literature review
[31]	Software evolution visualization: A systematic mapping study	Evolution	2013	SMS
[30]	Software evolution visualization techniques and methods - a systematic review	Evolution	2016	SLR
[12]	A Systematic Survey of Program Comprehension through Dynamic Analysis	Dynamic Analysis	2009	SLR
[23]	Information Visualization for Agile Software Development Teams	Process	2014	SMS

[1]SMS = Systematic Mapping Study, [2]SLR = Systematic Literature Review

IV. Results

The following results concern the final selection of 48 papers. Each sub-section provides an answer to a specific research question. We used the classification framework for questions 2 and 3. Further, the framework acts as a guide through the vastness of SV related field. Table I presents selected papers.

A. Publication Trends

First of all, the set of selected studies distribution is illustrated among publication years, research methodology and main topics covered.

1) Publication Years: Figure 2 presents the distribution of secondary paper publications on SV over the years. Despite the relatively small size of papers set and their nature, we can note that the scientific interest on SV in the last years has remained constant and steady. We found secondary studies from 2001 to 2018, and only 2004 had a discontinuity (peaks of 6 articles in 2009 and 2014). Thus, we observed a regular production of secondary in SV domain.

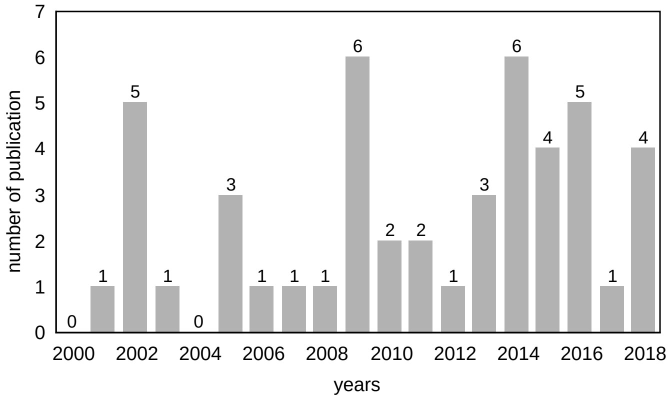

Fig. 2. Publication of SV selected reviews over the years (2000-2018).

2) Type of Publication: Figure 3 presents selected studies classified by their research types (or strategy). The term "Taxonomy" includes extension of taxonomies [13], [22]. We separated taxonomies and classifications because of their definition difference. The fundamental difference is that taxonomies describe relationships between items while classifications group the items. Systematic literature review and systematic mapping study represent an essential part of our set. As explained previously, reliable results obtained in those studies are necessary to build our tertiary systematic review.

3) Publication Main Topics: Figure 4 presents the topics that appear in the selected paper set. Some subjects remain popular in the research area, as it is the case for **Education, Architecture or Evaluation of developed tools**. Publication topics spread across different areas related to software engineering, and assess the breadth of SV. However, 37.5% of articles (18/48) are "General" SV topic, so they do not focus on a specific SV topic.

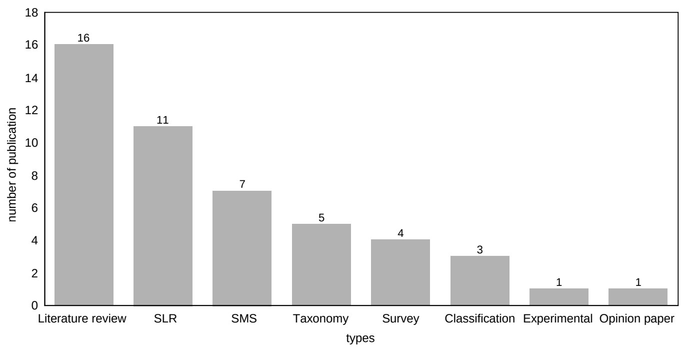

Fig. 3. Types of selected reviews publication.

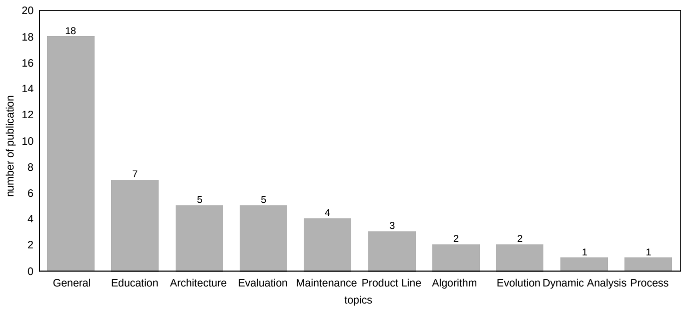

Fig. 4. Topics of selected reviews.

B. Purpose of Software Visualization and Corresponding Techniques Used

To define addressed subjects in this section, we chose in Figure 4 the best-represented subjects, that is *Education, Architecture, Maintenance, Product Line and Evolution*. Papers from "General" and "Evaluation" categories do not represent a software engineering application field, that is why they are not retained in this section. However, we exploit them in Section IV-C, dealing with issues regarding SV research field.

As expected, SV covers broad domains and involve different aspects of software engineering, from requirement engineering to programming activity. In their paper, Mattila et al. [34] found that the main goals of SV systems are software structure, behaviour and evolution understanding that uphold the broadness identified in our framework. The most represented field in our set is programming education [2], [4], [5], [9], [15], [18], [22], [24], [36], [38]. We noticed that some papers did not fit in defined categories, as they do not always focus on an application field. It is the case for reviewing tool papers, evaluation methods [21], [29], [35], opinion papers [37] or papers dealing with SV as a whole [33], [34]. Besides, we extracted data from those offering shared ideas on particular discussed aspects. Some data set fields were not represented in the framework if we did not find enough shared ideas allowing the presentation of an interesting state of the art. It especially is the case for team management, in which SV can help to communicate and to share knowledge. In what follows, we develop each category, focusing on corresponding SV system goals, commonly used techniques and audience.

1) Software Architecture Visualization: The first aspect that fits in software development activities we choose to focus on is software architecture visualization (SAV). In fact, this equally concerns the development, evolution and maintenance of software, but SV papers usually deal with software architecture (SA) as a whole. Indeed, architecture is a vital step towards building and maintaining software systems. It attracts developers, testers, project managers and even customers [2], [7], [11]. The goal is to find a meaningful and useful mapping scheme between the SA elements and visual metaphors. Architects need to realize the different characteristics of the architecture they design as complexity, coupling, cohesion and other attributes [7]. Visualization techniques applied in SAV are graph-based, notation-based as well as metaphors [7], [11]. They involve a multiplicity of views, data source visualization, classic navigation (following Schneiderman's mantra, see II).

2) Programming Education: This domain gathers a various audience, comprised of instructors, students as well as researchers. In programming education, the aim is to improve the learning of visualized concepts. Algorithm visualization (AV) and program visualization (PV) are used to fulfill this goal. PV helps students understand the run-time behaviour of other programs, acting as an educational tool to complement lectures, while AV brings algorithms to life by graphically representing their various states and animating the transitions between those states [2], [15]. One of the principal findings among programming education is that higher engagement level leads to higher educational benefits [22]. Nap et al. [22] found that students use AV technology has a greater impact on effectiveness than what AV technology shows them, it has to enable students to construct their understanding through active learning [2].

Despite effectiveness assessment issues that are furthered explored in discussion Section V, programming education is met with encouraging results. Urquiza-Fuentes et al. [5] verified that learning could be enhanced with PA visualizations in several ways. Improvements concerns knowledge acquisition at any engagement level, attitude towards materials or subject matters, and programming skills [5]. Regarding techniques, controlled viewing is thus suggested, and is, in fact, the most common form of direct engagement in these SV systems. Abstract two-dimensional graphics are also prevailing. [4].

3) Software Visualization and Maintenance: SV techniques are widely used in software maintenance and related domains, like reverse engineering, re-engineering, fault detection, and refactoring [13]. Examples of tasks SV could help to perform are debugging, database migration, cost estimation, change impact analysis, and design recovery [32].

Software comprehension is crucial to perform maintenance tasks, as large amounts of complex data need to be understood, and interactions between software engineers and automatic analyses are required [32], [35]. Empirical studies have shown that maintenance programmers spend 50% of their time to understand the software to be changed [60], enhancing the need for visualization to ease the comprehension process.

The interesting point is that the vast majority of researchers (80%) believes SV to be necessary for software maintenance [32], even if challenges still need to be faced, as we see in Section V. They employ numerous techniques in this domain, the most famous being graph-based, textual and automatic generated UML [32]. An important design step is to choose the right interactions. Indeed, due to the limited perception of humans and the amount of information that is to be visualized in such systems, it is important to be able to select and display just the data of interest. Filters, multiples views, synchronization and navigation between them are examples of interactions to accomplish these goals [29]. In this domain, they utilize 3D visualizations to increase information density and integrate those multiple views (e.g. structure and behaviour views), facilitating the maintenance perception [17].

4) Software Evolution Visualization: Software evolution (SE) is an important topic in software engineering that generally deals with large amounts of data, as it involves looking at whole project histories. One of the main aspects of this domain is the building of theories and models that allow understanding the past and present, as well as predicting future properties related to software maintenance activities, and hence support software maintenance tasks [31]. Consequently, we already discussed key aspects in previous Sections IV-B1 and IV-B3, due to existing interlinks in the categories we defined.

Regarding visualized artifacts, as SE generally deals with large amounts of data, it originates from heterogeneous sources such as Source Code Management (SCM) repositories, source code, Issue Tracking Systems (ITS), mailing and project discussion lists [30], [33], [34]. The main goal is to see how source code files changes over time.

Graph and matrix based, hierarchical and geometric projections are the most used techniques, with frequent use of colors, and common interactions as selection, navigation and zoom [30], [31]. Perspectives of visualization developed are structural (showing packages, classes, methods), dependency (showing relationships among software modules) as well as inheritance perspectives (e.g. Lanza's Polymetric view [57]).

Timelines or animations are used less than graph visualization, which is surprising as studying evolution in software indeed relates to time dimension [34].

5) Software Product Line Engineering: As stated by Pleuss et al. [47]

> *"In product line engineering (PLE) a major challenge is to handle a complexity of artifacts. Visual and interactive techniques aim to reduce the cognitive complexity and support the developer during challenging PLE tasks like product configuration."*

Visualization tools provide support in domain engineering (DE) activities of requirements engineering (DRE) and design (DD), as well as corresponding application engineering (AE) activities, that is when definition of feature models occurs, and requirements for each product are captured (ARE) and analyzed (AD) [10].

These techniques mostly aim at visualizing a unique artifact, that is feature models. They consist of trees, graphs and bar diagrams, using colours to distinguish the different features [45]. Those are basic visualizations, which rarely offer navigation between artifacts, and corresponding tools seem to be ad hoc techniques from frameworks as Eclipse. We see in Section V impacts of such utilization.

C. Issues Regarding Software Visualization Application

In this section, we do not focus on particular sub-fields of SV, and consider recurrent limitations evoked in papers that did not fit previous framework categories. We detail recommendation corresponding to each issue in the discussion VI.

1) Research Prototypes Scale: Price et al. [3] raised an issue a long time ago, concerning the problem of scalability of 1993 currently developed SV tools. Designated as "toy programs", they do not fit industrial scope in terms of the range of program inputs. The advice at this time was to focus on a production scale system to possibly use SV in the industry.

However, this issue is still encountered in program visualization software for programming education, where tools tend to be short-lived research prototypes [4]. Software maintenance researchers confirm that scalability and dealing with **space and time complexity of techniques are major challenges for SV for software maintenance** [32]. In that respect, they expect SV tools to only be appropriate for small to medium size systems.

2) Tools Effectiveness Validation: One of the mainly discussed issues regarding SV tools in the research field is the lack of rigorous validation techniques. The issue recurs in numerous papers [3], [4], [34], [41]. In the discussion, we observe that each sub-field experience this issue and it has its guidelines to overcome it. The concerning fact is that Price et al. [3], in their 1993 taxonomy, already outlined the insufficiency in the empirical evaluation of prototypes effectiveness. In a more recent work, presenting a systematic review of SV as a whole, Mattila et al. [34] indicated the primary papers they reviewed to not be rigorous enough in the way they evaluate their work. Indeed, surveys or controlled experiments are not popular comparing to case studies, though they are described as being a proper validation process [41]. Beyond the idea of effectiveness, primary papers consistently forget to state research methods and questions [34]. This deficiency rigor is not justified, regarding SV field maturity.

3) Scope-related Vision: Reverse engineering, re-engineering and software maintenance professionals think that SV researches are too much metaphor related and could gain from focusing on the scope of the tool being developed [32]. Indeed, it is even truer in 3D visualization, where the aim of SV is not to create impressive images, but images that evoke viewer mental images for better software comprehension [17]. In the meantime, Petre [25] studied mental imagery and SV in high-performance software development team . The author suggested that the SV system should embody more knowledge of the application domain, to visualize software in context. Another problem of focusing too much on the techniques is the lack of usability. We should consider human factors in the design and evaluation of SV tools [13], [17].

V. Discussion

A. Application Domains of Software Visualization

In this section, we mention the main research issues, then present the means to address them. Most suggestions come from our paper set and for this reason, we recommend reading the corresponding quoted papers as they contain far more details and contextualized content. We often introduce future research gaps and recommendation for researchers to find guidance through this work.

1) Software Architecture Visualization: The challenge SV tools have to face in software architecture visualization (SAV) is the problem of scalability, since this is a field involving large amounts of data, and todays software systems are increasingly large and complex. Many solutions appear to meet this challenge. They consist in consideration of following issues in future researches.

First of all, Plesus *et al.* [48] discussed that SV tools do not provide good viewpoints, abstraction levels and filters needed to understand complex software architectures. One remedy could be to rely on visualization techniques which are able to amplify the human cognition process. The trend in recent papers is to use real metaphors (e.g. city metaphor) instead of abstract ones, to represent software artifacts [7].

Besides, being able to prove the effectiveness of SV systems developed is another way to face the complexity problem. Indeed, the evaluation of SAV still lacks rigor, as this is the case for many SV tools [7], [11]. It is essential to find criteria to determine what makes an effective architecture visualization, as well as paying attention to use more objective evaluation methods (e.g., controlled experiments) for providing convincing evidence to support its effectiveness [7], [11].

Another advice is to consider automation, as already mentioned by Price *et al.* in 1993 [3]. Shanin *et al.* [11] claim that automated and semi-automated tools permit to gather and provide higher levels of evidence for architecture visualization techniques, which also lower efforts produced by practitioners.

Regarding research gaps, there are few applied visualization techniques in architectural analysis, synthesis, implementation and reuse activities. Parallel coordinates and bundled diagram layouts are interesting techniques rarely explored in developed systems [44]. In popular activities like maintenance and comprehension, part of the growing area of research investigates the application of 3D graphics, with positive results [17].

We emphasize the conduct of industrial surveys as this could allow the understanding of how SA practitioners employ visualization techniques in the architecture process, and what issues prevent them from being adopted in SA [11].

2) On the Use of Algorithm Visualization in Education: Although results in this field are positive, as we found in Section IV-B2, most selected papers agree there is still a lack of careful evaluation on programming education SV. Indeed, Sorva *et al.* and Urquiza-Feuentes *et al.* [4], [5] speak about informal evaluations with little contributions to future improvements of evaluated systems.

The use of theoretical foundations such as the social constructivism theory evoked in Shahin *et al.* [11] is a running notion. Constructivism states that students actively construct knowledge rather than passively receiving and storing already made knowledge. Most of the effective AVs were built and guided by such cognitive theories, and considering other learning theories from different domains could result in important contributions in terms of effectiveness [9]. We also suggest for PV to use concrete visual allegories and gamification [24].

Furthermore, the nature of some developed tools is a worrying matter in this field: they are often short-lived research prototypes, suffering a low adoption rate [4], [9]. Back in 2002, Hundhausen *et al.* [2] stated that *"visualization technology has failed to catch on in the mainstream computer science education"*. In 2014, Isohanni *et al.* [18] qualified SV tools use in class as seldom, besides often appearing to deal with only basic programming, data structures and algorithms.

Besides, contrary to recommendation usages, they are commonly used by teachers and not students, for demonstrations during classroom lectures or other strategies involving learners in passive interactions [18], [22]. Finally, the community development around AVs is an interesting idea that appears in our set, gathering various stakeholders such as developers, educators, researchers, and end-users. A collaborative effort would allow small-scale AVs developers to benefit from access to existing AV implementations, rating system for existing AVs, comments and feedback [15].

We conclude on programming education by pointing out the fact that there is a growing area of research investigating the application of 3D graphics and algorithm animation for educational purposes [17].

3) Software Maintenance Visualization: The concerns of current SV tools for maintenance is that they do not overcome the domain specific issues of scalability and complexity, and are only appropriate for small to medium size systems [32]. Creating high abstraction levels and different level of details in tools developed could be a solution, as maintenance programmers work at every level of abstraction [29]. 3D visualizations are developed to present different abstraction levels using source code, object-oriented systems or software architectures [17] .

The nature of researches is another critical concern in software maintenance visualization, which is disconnected from domain problems. While designing SV tools, features have to be scope related rather than visualization related [32]. A thorough analysis of dedicated applications is necessary to correspond to real life usages.

Clone visualizations can be considered as a sub-field helping refactoring activities. However, none of the existing visualizations use an inheritance structure or a run-time interaction of program units which are essential for assessing refactoring opportunities [6].

Finally, we recommend to enhance community interactions. Indeed, SV could gain from collaboration with graph drawers, for example, to benefit from their expertise, planarization techniques to reduce graph visualization complexity while dealing with large systems [32]. Tools should be available for the research community, enhancing their customization and composition. Doing this would support standard exchange formats and benefits to all involved stakeholders.

Moreover, for those willing to develop SV tools focusing on maintenance tasks, we recommend the reading of the work of Kienle and Muller [29] who identified requirements (quality attributes and functional requirements) for SV tools.

4) Software Evolution: Bani-Salameh *et al.* [30] point out the decreasing number of published studies in SE visualizations. Is the area failing to reach its goals? The fact that SE visualization approaches do not always focus on achieving practical SE goals, besides presenting partially validated systems could explain such disinterest. Those are close issues to what software maintenance field face, as they are interlinked. Indeed, we noticed a lack of cooperative and comparative work as it was the case in the previous Section V-A3.

About research gaps, it would be interesting to investigate if SE could be visualized in ways that emphasizing the time dimension and even use similar visualization methods than in software execution visualization [34].

5) Software Product Line Engineering: Most approaches do not tap on the wealth of tooling and visualization techniques currently available. Instead, they use either ad-hoc tools and techniques (e.g. basic graphics APIs provided by the Java SDK.49) or are based on development frameworks like Eclipse Modeling Framework [10], [45]. Indeed, researchers did not employ more advanced visualization techniques or tools, just stand-alone tools that are not integrated. Pleuss *et al.* [47] stated that *"the full potential of visualization in the context of PLE has not been exploited so far"*.

Even in the use of colours to distinguish the artifacts that belong to each feature, a scalability problem appears as thousands of features are to be represented. More experiments with larger scopes are needed, as well as the integration of tools developed in popular IDEs, and the usage of more advanced visualization techniques and tools [10]. We can thus see that handling scalability and multivariate data visualization are two open challenges with a high impact potential in the PLE community.

Numerous works used industrial or academic examples, which is a good indication of the importance of visualization techniques in industrial settings and attention research community is putting on the domain.

In this section, we recommend to enhance community interactions. Indeed, SV could gain from collaboration with graph drawers, for example, to benefit from their expertise, planarization techniques to reduce graph visualization complexity while dealing with large systems [32]. Tools should be available for the research community, enhancing their customization and composition. Doing this would support standard exchange formats and benefits to all involved stakeholders.

VI. Recommendations

We provide advice gathered from the different analyzed search papers. Among all recommendations identified in our paper set, we choose to regroup those appearing in general recommendations. They are presented in Table II, allowing to highlight recommendations that appear the most. The merging of such reliable results produces sound guidelines for future research or development in the SV field.

A. General

We can see in Table II (R4), that community development in SV is recommended by 21% of our paper set. Indeed, the gathering of different stakeholders as developers, researchers or educators would enhance access to previous SV implementations, feedback and rating systems.

Our recommendations for this community gathering to happen are the following:

- For SV projects, creating a website that presents contributors, developed tool and provide a downloading link (in the case of open source project) would permit various stakeholders to interact with it.
- Any of these webpages could be submitted to websites or blogs referencing them, an example is Craig Anslow's blog (https://softvis.wordpress.com/about/), which is currently active. Gathering projects is more convenient for the different stakeholders to find them, avoiding the browsing of numerous webpages.

B. Developers

While designing a SV, the two main recommendations, according to Table II, are to identify real software engineering tasks the tool will support (R3) and provide details on demand as an interaction (R2), especially while working on complex systems, to avoid cognitive overload. It depends on which subfield we are working on, but those recommendations are shared by respectively 21 and 23% of our set, placing them at the level of sound guidelines.

C. Instructors

Teachers mostly use AV and PV tools as a means of teaching in the education field as discussed in Section V-A2.

Here, we recommend as 14.5% of the selected articles, to promote active learning for students (R5).

D. Researchers

There is a crucial need to conduct empirical studies to existing visualization (R1, shared by 40% of our set), or at least to think about such validation in the design of future visualization systems.

We recommend reading previous work that only focus on this aspect, the first being Sensalire *et al.* [21]. It provides guidelines for tool and task selection, choosing and training of participants, and analyzing the results from the evaluation conducted. Merino *et al.* provide guidelines to increase the evidence of the effectiveness of SV approaches, thus improving their adoption rate [8]. Regarding PV and AV tools, Urquiza-Fuentes *et al.* [5] recommend narrative and textual contents, feedback to students answers, and a student-centered approach in the design of PAV construction kits .

VII. Threats to Validity

SLR process can hardly be fully comprehensive, as some papers can slip through the internet. Data extraction is often biased because impartiality is hard to meet, especially without previous experiences in the domain. We try to follow as strictly as possible all good practices from guidelines readings and try to be as objective as possible. Thus, we describe the main threats to validity to our study and how we mitigated them.

A. Construct Validity.

We mitigated this threat by searching the studies on different data sources. However, research studies that did not contain the term "software visualization" were not added in our study. However, we also argue that our string requests provide relevant papers that could efficiently answer in our research questions. We added a selection layer by using inclusion and exclusion criteria.

B. Internal Validity.

We mitigated this threat by both reading and sharing our interpretation of each read papers, thus mitigating the possibility of misinterpreting them. We also extracted and aggregated the main ideas and recommendation in Section V and Section VI which minimize the impact of a misinterpreted paper.

C. External Validity.

The main external threat to our study is that we excluded selected papers published before 2000 except for Price *et al.* [3]. We argue that the missing papers do not significantly impact our review, as the present ones already share similar ideas and recommendations, in addition to be more topical.

D. Conclusion Validity.

Reproducibility of our study could be obstructed due to a bias in the data collection. By establishing a data extraction protocol that we rigorously follow, we mitigated this threat by maintaining a set of spreadsheets to keep records but also to reorganize our data set. This also allowed us to identify any anomalies in our set.

VIII. Related Work

To the best of our knowledge, our work is the first tertiary review on the subject of SV. However several studies have used systematic mapping study process to propose state of the art these are part of the selected set of data used in this work. Mattila *et al.* [34] use a systematic review process to generate findings on SV as a whole. Hence, the following studies also discussed secondary articles, although they are not tertiary studies. Novais *et al.* [31] offer a systematic mapping study on software evolution visualization. Survey on software architecture visualization is presented by Carpendale and Ghanam [7]. Caserta and Zendra [46] present the findings

TABLE II
SHARED RECOMMENDATIONS IDENTIFIED IN OUR SET

	Statement	Description	References	Total
R1	Conduct empirical studies to validate usefulness	On existing visualizations, or the ones being developed, to add them values and speed up the integration process. Controlled experiment, unbiased subjects, quantitative measures	[1]–[3], [7], [8], [10], [11], [21], [24], [26], [27], [30], [31], [33], [39], [41], [43], [44], [46], [47]	39.5% (19/48)
R2	Provide details on demand, avoid cognitive overload	Support human cognition, provide detail-on-demand interaction or higher level of abstraction	[6], [7], [13], [16], [20], [25], [28], [39], [44], [47], [48]	23% (11/48)
R3	Map techniques to meet specific goals, real problems	Embodying more knowledge of the application	[12], [15], [25], [31], [32], [37], [40], [44], [47], [48]	21% (10/48)
R4	Think about interoperability, community collaboration	Through exchange format, making tools available online (2, AV), reuse and use of recent techniques	[9], [14], [17], [26], [28], [30], [32], [35], [44], [45]	21% (10/48)
R5	Engage learner in activities	Allow them to construct their own visualization (active learning), student-centered vision	[1], [2], [4], [5], [9], [18], [22]	14.5% (7/48)
R6	Think about usability in the design	Keep interaction simple, provide help system	[5], [17], [20], [28], [35], [37]	12.5% (6/48)
R7	Have strong theoretical foundation	Psychology theories, other than constructivism for example	[1]–[3], [9], [38]	10.5% (5/48)
R8	Use multiple view	Ease selection of data, through source code view or more abstract ones	[3], [19], [20], [26], [32]	10.5% (5/48)
R9	Use automated tools	Reduce efforts to use the visualization	[1]–[3], [11]	8% (4/48)
R10	Exploit Virtual Environment	To extend visualization to perception, ease collaborative SE process, increase amount of data shown	[17], [32], [36], [46]	8% (4/48)
R11	Consider real needs of viewer	Meet stakeholders requirements, understand viewer objectives	[8], [26], [32], [44]	8% (4/48)
R12	Scale up to handle complexity of current software	Handle large amount of data, on production scale level	[10], [28], [37]	6% (3/48)
R13	State research method and questions	Besides, discuss the approach goals and validation strategy	[30], [33], [40]	6% (3/48)
R14	Enable customization	Permits to meet viewer-specific needs	[28], [37]	4% (2/48)
R15	Use Gamification	Contribute to effectiveness of learning tools	[9], [24]	4% (2/48)

of a far-reaching literature survey on visualizing static aspects and the evolution of the software.

IX. CONCLUSION

Out of the 212 papers extracted, we reviewed 48 papers related to SV and its sub-fields, and present them in Table I. Our first research question [RQ1] provides insights on the state of SV research area regarding secondary papers. It appeared to be enough active to permits the production of our work. We found which sub-fields are the most active (those addressed in Section V).

The second one, [RQ2], allowed us to identify the purpose and most used techniques of chosen sub-fields, while we evoke corresponding issues in the discussion.

Finally, the last research question [RQ3] permits to emphasize persistent issues encountered in the research area which is slowing SV tools integration in the industry.

Thus, we resumed main findings among popular sub-fields of SV, identified corresponding issues research is facing and provide future search suggestions by gathering main recommendation in Table II. Indeed, SV tool evaluations remain superficial in most of the evoked fields, the lack of connections between tool designs and their scope remains in most sub-fields, as well as the dispersal of the research community.

REFERENCES

[1] Gallagher, K., Hatch, A., and Munro M. ”Framework for software architecture visualization assessment”, *Third IEEE Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, 2005, 10.1109/VISSOF.2005.1684309.

[2] Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko ”A Meta-Study of Algorithm Visualization Effectiveness”, *Journal of Visual Languages & Computing*, Volume 13, Issue 3, Pages 259-290, 2002, https://doi.org/10.1006/jvlc.2002.0237.

[3] Price, B.A., Baecker, R.M., and Small, I.S. ”A Principled Taxonomy of Software Visualization”, *Journal of Visual Languages and Computing* , Volume 4, Issue 3, Pages 211-266, 1993.

[4] Sorva, J., Karavirta, V., and Malmi, L. ”A review of generic program visualization systems for introductory programming education”, *ACM Transactions on Computing Education* , Volume 13, Issue 4, Article 15, 2013, 64 pages, http://dx.doi.org/10.1145/2490822.

[5] Urquiza-Fuentes, J. and Ángel Velázquez-Iturbide, J. ”A survey of successful evaluations of program visualization and algorithm animation systems”, *ACM Transactions on Computing Education*, Volume 9, Issue 2, Article 9, 2009, 21 pages, 10.1145.1538234.1538236.

[6] Basit, Hamid, Hammad, Muhammad, and Koschke, Rainer. ”A survey on goal-oriented visualization of clone data”, *Third IEEE Working Conference on Software Visualization (VISSOFT)*, Pages 46-55, 2015, 10.1109/VISSOFT.2015.7332414.

[7] Y. Ghanam and S. Carpendale, ”A Survey Paper on Software Architecture Visualization”, 2008.

[8] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz, ”A systematic literature review of software visualization evaluation”, *Journal of Systems and Software*, Volume 144, Pages 165-180, 2018, https://doi.org/10.1016/j.jss.2018.06.027.

[9] Abdullah Al-Sakkaf, Mazni Omar, and Mazida Ahmad ”A systematic literature review of student engagement in software visualization: a theoretical perspective”, *Computer Science Education*, Volume 29, Issue 2-3, Pages 283-309, 2019 10.1080/08993408.2018.1564611.

[10] LopezHerrejon RE, Illescas S,and Egyed A. ”A systematic mapping study of information visualization for software product line engineering”, *Journal of Software: Evolution and Process*, Volume 30, Issue 2, 2018, https://doi.org/10.1002/smr.1912.

[11] Shahin, M., Liang P., and Ali Babar M. ”A systematic review of software architecture visualization techniques”, *Journal of Software and Systems*, Volume 94, Pages 161-185, 2014, http://dx.doi.org/10.1016/j.jss.2014.03.071.

[12] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, ”A Systematic Survey of Program Comprehension through Dynamic Analysis”, *IEEE Transactions on Software Engineering*, Volume 35, Issue 5, Pages 684-702, 2009, 10.1109/TSE.2009.28.

[13] J. I. Maletic, A. Marcus and M. L. Collard, ”A task oriented view of software visualization”, *Proceedings First International Workshop on Visualizing Software for Understanding and Analysis*, Pages 32-40, 2002, 10.1109/VISSOF.2002.1019792.

[14] Cecile Yehezkel ”A taxonomy of computer architecture visualizations”, *Proceedings of the 7th annual Conference on Innovation and Technology in Computer Science Education*, Volume 34, Issue 3, Pages 101-105, 2002, http://dx.doi.org/10.1145/544414.544447.

[15] Shaffer, C. A., Cooper, M. L., Alon, A. J. D., Akbar, M., Stewart, M., Ponce, S., and Edwards, S. H. ”Algorithm visualization: The state of the field”, *ACM Transactions on Computing Education*, Volume 10, Issue 3, Article 9, 2010, 10.1145/1821996.1821997.

[16] Vasconcelos, Renan, Marcelo Schots and Cludia Maria Lima Werner An information visualization feature model for supporting the selection of software visualizations, *22nd International Conference on Program Comprehension*, 2014.

[17] A. R. Teyseyre and M. R. Campo ”An Overview of 3D Software Visualization,” *IEEE Transactions on Visualization and Computer Graphics*, Volume 15, Issue 1, Pages 87-105, 2009, 10.1109/TVCG.2008.86.

[18] Essi Isohanni and Hannu-Matti Jrvinen. ”Are visualization tools used in programming education?: by whom, how, why, and why not?”, *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, Pages 35-40, 2014, https://doi.org/10.1145/2674683.2674688.

[19] Mariam Sensalire, Patrick Ogao, and Alexandru Telea ”Classifying desirable features of software visualization tools for corrective maintenance”, *Proceedings of the 4th ACM symposium on Software visualization*. Pages 87-90, 2008, https://doi.org/10.1145/1409720.1409734.

[20] R. Tilley, Scott and Huang, Shihong. ”Documenting software systems with views III: Towards a task-oriented classification of program visualization techniques”, *Proceedings of ACM Annual International Conference on Computer Documentation*. Pages 226-233, 2002, 10.1145/584955.584988.

[21] Sensalire, M., Ogao, P., and Telea, A. ”Evaluation of Software Visualization Tools: Lessons Learned”, *5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2009, 10.1109/VISSOF.2009.5336431.

[22] Thomas L. Naps, Guido Rling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J. ngel Velzquez-Iturbide. ”Exploring the role of visualization and engagement in computer science education”, *Working group reports from Innovation and Technology in computer Science Education*. Pages 131-152, 2002, http://dx.doi.org/10.1145/782941.782998.

[23] J. Paredes, C. Anslow and F. Maurer ”Information Visualization for Agile Software Development”, *Second IEEE Working Conference on Software Visualization*, Pages 157-166, 2014, 10.1109/VISSOFT.2014.32.

[24] J. Hidalgo-Cspedes, G. Marn-Ravents and V. Lara-Villagrn ”Learning principles in program visualizations: A systematic literature review”, *IEEE Frontiers in Education Conference*, 2016, Pages 1-9, 10.1109/FIE.2016.7757692.

[25] Marian Petre ”Mental imagery and software visualization in high-performance software development teams”, *Journal of Visual Languages & Computing*, Volume 21, Issue 3, Pages 171-183, 2010, https://doi.org/10.1016/j.jvlc.2009.11.001.

[26] Storey, M. A. D., ubrani, D., and German, D. M. ”On the use of visualization to support awareness of human activities in software development: a survey and a framework”, *Proceedings of the 2005 ACM symposium on Software visualization*, Pages 193-202, 2005, 10.1145/1056018.1056045.

[27] Mller, Richard, and Zeckzer, Dirk. ”Past, Present, and Future of 3D Software Visualization: A Systematic Literature Analysis”, *6th International Conference on Information Visualization Theory and Applications*, Pages 63-74, 2015, 10.5220/0005325700630074.

[28] Ghaleb, Taher, Alturki, Musab, and Aljasser, Khalid. ”Program comprehension through reverse-engineered sequence diagrams: A systematic review”, *Journal of Software: Evolution and Process*, Volume 30, Issue 11, 2018, 10.1109/VISSOF.2007.4290693.

[29] H. M. Kienle and H. A. Muller ”Requirements of Software Visualization Tools: A Literature Survey”, *4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, Pages 2-9, 2007.

[30] Bani-Salameh, Hani, Ahmad, Ayat, and Aljammal, Ashraf. ”Software evolution visualization techniques and methods - a systematic review” *7th International Conference on Computer Science and Information Technology*, Pages 1-6, 2016, 10.1109/CSIT.2016.7549475.

[31] Renato Lima Novais, Andr Torres, Thiago Souto Mendes, Manoel Mendona, Nico Zazworka, ”Software evolution visualization: A systematic mapping study”, *Information and Software Technology*, Volume 55, Issue 11, 2013, Pages 1860-1883, https://doi.org/10.1016/j.infsof.2013.05.008.

[32] Koschke, Rainer. ”Software Visualization in Software Maintenance, Reverse Engineering, and Reengineering: A Research Survey”, *Journal on Software Maintenance and Evolution*. Volume 15, Pages 87-109, 2003, 10.1002/smr.270.

[33] D. Graanin, K. Matkovi, and M. Eltoweissy. ”Software visualization”, *Innovations in Systems and Software Engineering*, Volume 1, Issue 2, Pages 221-230, 2005, https://doi.org/10.1007/s11334-005-0019-8.

[34] A. Mattila, P. Ihantola, T. Kilamo, A. Luoto, M. Nurminen, and H. Vtj. ”Software visualization today: systematic literature review”, *Proceedings of the 20th International Academic Mindtrek Conference*, Pages 262-271, 2016, https://doi.org/10.1145/2994310.2994327.

[35] S. Bassil and R. K. Keller ”Software visualization tools: survey and analysis”, *Proceedings 9th International Workshop on Program Comprehension*, Pages 7-17, 2001, 10.1109/WPC.2001.921708.

[36] Libert, Cdric and Vanhoof, Wim. ”Survey of Software Visualization Systems to Teach Message-Passing Concurrency in Secondary School”, *Highlights of Practical Applications of Cyber-Physical Multi-Agent Systems: International Workshops of PAAMS 2017*, Pages 386-397, 2017, 10.1007/978-3-319-60285-1_33.

[37] S. P. Reiss ”The Paradox of Software Visualization,” *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, Pages 1-5, 2005, 10.1109/VISSOF.2005.1684306.

[38] Abdullah Al-Sakkaf, A., Omar, M., and Ahmad, M. ”Theoretical Underpinnings of Learner Engagement in Software Visualization System: A Systematic Literature Review Protocol”, *International Journal of Engineering & Technology*, Volume 7, Issue 3.20, Pages 35-39, 2018 http://dx.doi.org/10.14419/ijet.v7i3.20.18727.

[39] Xu, Yangyang, Liu, Yan, and Zheng, Jiabin. ”To Enlighten Hidden Facts in The Code: A Review of Software Visualization Metaphors”, *27th International Conference on Software Engineering and Knowledge Engineering*, Pages 294-297, 2015, 10.18293/SEKE2015-203.

[40] L. Merino, M. Ghafari and O. Nierstrasz ”Towards Actionable Visualisation in Software Development”, *IEEE Working Conference on Software Visualization*, Pages 61-70, 2016, 10.1109/VISSOFT.2016.10.

[41] A. Seriai, O. Benomar, B. Cerat and H. Sahraoui, ”Validation of Software Visualization Tools: A Systematic Mapping Study,” *Second IEEE Working Conference on Software Visualization*, Pages 60-69, 2014, 10.1109/VISSOFT.2014.19.

[42] Balogh, Gergo. ”Validation of the City Metaphor in Software Visualization”, *Computational Science and Its Applications*, Pages 73-85, 2015, 10.1007/978-3-319-21413-9_6.

[43] Sulr, M., Bakov, M., Chodarev, S., and Porubn, J. ”Visual augmentation of source code editors: A systematic mapping study”, *Journal of Visual Languages & Computing*, Volume 49, Pages 46-59, 2018, 10.1016/j.jvlc.2018.10.001.

[44] Khan, Taimur, Barthel, Henning, Ebert, Achim, and L liggesmeyer, Peter. ”Visualization and Evolution of Software Architectures”, *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering*, Volume 27, 2012, 10.4230/OASIcs.VLUDS.2011.25.

[45] R. E. Lopez-Herrejon, S. Illescas and A. Egyed ”Visualization for Software Product Lines: A Systematic Mapping Study”, *IEEE Working Conference on Software Visualization*, Pages 26-35, 2016, 10.1109/VISSOFT.2016.11.

[46] Pierre Caserta, and Olivier Zendra. ”Visualization of the Static aspects of Software: a survey”, *IEEE Transactions on Visualization and Computer Graphics, Institute of Electrical and Electronics Engineers*, Volume 17, Issue 7, Pages 913-933, 2011, 10.1109/TVCG.2010.110.

[47] Andreas Pleuss, Rick Rabiser, and Goetz Botterweck. ”Visualization techniques for application in interactive product configuration”, *Proceedings of the 15th International Software Product Line Conference*, Volume 2, Article 22, 8 pages, 2011, http://dx.doi.org/10.1145/2019136.2019161.

[48] Lemieux, Franois and Martin Salois. Visualization Techniques for Program Comprehension - A Literature Review. *5th International Conference on Software Methodologies, Tools and Techniques*, 2006.

[49] S. Diehl ”Software visualization: visualizing the structure, behaviour, and evolution of software”, *Springer Science & Business Media*, 2007.

[50] Stuart T. Kard, Jock D. Mackinlay, and Ben Scheiderman ”Readings in Information Visualization, Using vision to think”, 1999.

[51] Petersen, K., Vakkalanka, S., and Kuzniarz, L. ”Guidelines for conducting systematic mapping studies in software engineering: An update”, *Information and Software Technology*, Volume 64, Pages 1-18, 2015.

[52] B. Schneiderman, ”The Eyes Have It : A Task by Data Type Taxonomy for Information Visualizations”, *Proceedings 1996 IEEE Symposium on Visual Languages*, 1996, 10.1109/VL.1996.545307.

[53] T. Dyba, T. Dingsoyr, and G.K. Hanssen, ”Applying systematic reviews to diverse study types: an experience report”, *First International Symposium on Empirical Software Engineering*, 2007, 10.1109/ESEM.2007.59.

[54] M. Kuhrmann , D. Mndez Fernndez and M. Daneva ”On the pragmatic design of literature studies in software engineering: an experience-based guideline”, *Empirical Software Engineering*, Volume 22, Issue 6, Pages 2852-2891, 2017, 10.1007/s10664-016-9492-y

[55] Wohlin, C., Runeson, P., da Mota Silveira Neto, P.A., m, E.E., do Carmo Machado, I., and de Almeida, E.S. ”On the reliability of mapping studies in software engineering”, *Journal of Systems and Software* , Volume 86, Issue 10, Pages 2594-2610, 2013, http://dx.doi.org/10.1016/j.jss.2013.04.076

[56] M. V. Zelkowitz An update to experimental models for validating computer technology, *Journal of Systems and Software*, Volume 82, 2009, https://doi.org/10.1016/j.jss.2008.06.040.

[57] M. Lanza, S. Ducasse ”Polymetric views a lightweight visual approach to reverse engineering”, *IEEE Transactions on Software Engineering* Volume 29, Issue 9, Pages 782-795, 2003, 10.1109/TSE.2003.1232284.

[58] Gilmore DJ and Green TRG, ”Comprehension and recall of miniature programs”, *Journal of Man- Machine Studies*, Volume 21, Pages 3148, 1984, 10.1016/S0020-7373(84)80037-1.

[59] Roman, G.-C. and Cox, K. C. ”A Taxonomy of Program Visualization Systems”, *IEEE Computer*, Volume 26, Issue 12, Pages 11-24, 1993, 10.1109/2.247643.

[60] Fjeldstadt RK and Hamlen WT ”Application program maintenance study: Report to our respondents”, *Proceedings of the GUIDE 48. IEEE Computer Society Press*, 1984.