Towards Actionable Visualisation in Software Development

Leonel Merino, Mohammad Ghafari, and Oscar Nierstrasz
Software Composition Group, University of Bern
Bern, Switzerland
{merino, ghafari, oscar}@inf.unibe.ch

Abstract—Although abundant studies have shown how visualisation can help software developers to perform their daily tasks, visualisation is still not a common practice since developers have little support for adopting a proper visualisation for their needs.

In this paper we review the 346 papers published in SOFTVIS/VISSOFT venues and identify 65 design study papers that describe how visualisation is used to alleviate various difficulties in software development. We classify these studies into several problem domains that we collected from the research on software development community, and highlight the characteristics of each study. On the one hand, we support software developers to put visualisation in action by mapping existing techniques to particular needs in various problem domains. On the other hand, we help researchers in the field by exposing domains with little visualisation support. We found a disconnect between the problem domains on which visualisation have focused and the domains that get the most attention from practitioners.

I. INTRODUCTION

Software visualisation provides enormous advantages for development; to name a few, it supports project managers in communicating insights to their teams [1], it guides testers when exploring code for anomalies [2], it helps analysts to make sense of multivariate data [3], and it aids new developers in open software communities [4]. However, visualisation is not yet commonly used by developers. More than a decade ago researchers wondered why is software visualization not widely used? [5]. They realised one of the reasons is that efforts in software visualisation are out of touch with the needs of developers [6]. Several attempts have tried to fill in the gap and encourage developers to adopt visualisation. For instance, Maletic et al. [7] proposed a taxonomy of software visualisation to support various tasks during software development; Schots et al. [8] extended this taxonomy by adding the resource requirements of visualisations, and providing evidence of their utility; Storey et al. [9] proposed a framework to assess visualisation tools; Kienle et al. [10] performed a literature survey to identify quality attributes and functional requirements for software visualisation tools; Padda et al. [11] proposed some visualisation patterns to guide users in understanding the capabilities of a given visualisation technique; and Sensalire et al. [12] classified the features that users require in software visualisation tools. However, the lack of organisation among visualisation approaches is still an important barrier to finding and using them in practice [8]. In fact, developers are still unaware of existing visualisation techniques to adopt for their particular needs. A few studies have tried to address this issue by investigating to which software engineering tasks particular visualisation techniques have been applied [13]–[15]. Nevertheless, we believe these studies are still coarse-grained to match a suitable visualisation to their concrete needs.

When developers perform a particular programming task they ask some questions such as “what code is related to a change?” or “where is this method called?”. Several studies have investigated such questions and classified them into groups [16]–[18]. Indeed, such questions reflect developer needs, and we believe mapping them to existing types of visualisation can help developers to adopt visualisation in their daily work. Our twofold goal is 1) to help practitioners to find suitable visualisation for their specific needs, and 2) to assist researchers in the field to identify problem domains with little visualisation support. Accordingly, we focus on the following research questions:

RQ1. What are the characteristics of visualisation techniques that support developers needs?
RQ2. How well are various problem domains supported by visualisation?

We reviewed relevant literature in the software visualisation field and found that one third of the studies combined various visualisation techniques, but most of them belong to one of the following two types: 1) techniques that use geometric transformations to explore structure and distribution, and 2) pixel-oriented techniques that are suitable for displaying large amounts of data. We found extensive visualisation support for needs in the domains of history, dependency, and performance, whereas there is little support for needs in rationale, refactoring, and policies domains.

The remainder of the paper is structured as follows: Section II describes the methodology that we followed to collect relevant literature and select design studies proposed in the software visualisation field; Section III presents our results by classifying them based on their task, need, audience, data source, representation, tool, and medium [7]; Section IV discusses our research questions and threats to validity of our findings, and Section V concludes and presents future work.
II. METHODOLOGY

We applied the Systematic Literature Review (SLR) approach, a rigorous and auditable research methodology for Evidence-Based Software Engineering (EBSE). The method offers a means for evaluating and interpreting relevant research to a topic of interest. We followed Keele’s comprehensive guidelines [19], which make it less likely that the results of the literature survey will be biased.

A. Data sources and search strategy

We sought papers that are relevant to the aim of this study, i.e., to propose a visualisation technique which demonstrated to be useful to solve a specific problem in software development. Although such papers are expected to be found across multiple software engineering venues, we decided to collect them from the complete set of papers published by SOFTVIS [20] and VISSOFT [21]. We opted for these two venues because we believe their fourteen editions and hundreds of papers dedicated specially to software visualisation offer a sound body of literature reflected in the good classification that they obtain in the CORE ranking [22] (that considers citation rates, paper submission and acceptance rates among other indicators). Figure 1 summarises the number of papers collected as well as those included in this study.

![Figure 1. Collection of 346 papers published in SOFTVIS/VISSOFT venues.](image)

B. Included and excluded studies

We searched for problem-driven studies in which we could identify the role of the user, specific development needs, a proposed visualisation technique, and an evaluation demonstrating utility. We excluded short papers of one or two pages (like posters, keynotes and challenges) which due to limited space are unlikely to contain enough detail. We also excluded short papers for which a longer version exists. Of the 273 remaining papers we selected design study papers that describe how a visualisation is suitable for tackling a particular problem in software development. We included such papers in our study and excluded papers in the other categories proposed by Munzner [23] (technique, system, formalism, and model) because we considered them unlikely to provide a visualisation to tackle a problem in software development.

We classified the types of papers by first reading the abstract, second the conclusion, and finally, in the cases where we still were not sure of their main contribution, reading the rest of the paper. Although some papers might exhibit characteristics of more than one type, we classified them focusing on their primary contribution. Figure 2 shows the outcome of our classification. We identified 65 design study papers and included them in the study. Although approximately two thirds of the papers came from VISSOFT, selected papers that we classify as design studies are balanced.

![Figure 2. Classification of the 273 SOFTVIS/VISSOFT papers by type.](image)

Figure 3 shows a stacked line-chart with the evolution of the number of papers published in the venues by type. Although all types show an upward trend, the most notable cases are system, model and formalism types. Instead, design papers increased moderately. Traditionally, the number of papers in SOFTVIS editions (2003-2010) was consistently higher than in VISSOFT workshops (2002-2011). The trend of the publications once they merged in the VISSOFT conferences (2013-2015) seems more influenced by SOFTVIS.

![Figure 3. Evolution of the number of papers published in SOFTVIS/VISSOFT.](image)

Figure 4 shows a visualisation of the universe of 346 papers published in SOFTVIS/VISSOFT. In this visualisation, rectangles represent papers, their height encodes the number of pages (a 5-page paper is depicted by a square), and the colour is used to identify its venue (VISSOFT in blue, and SOFTVIS in red). We used the intensity of the colour to represent the publication year, thus the darker the colour the newer the paper. Edges connect authors (grey circles) to papers (rectangles). The 65 selected design study papers are distinguished by a black border and a label on top. In the visualisation the topology of the community is exposed. A few large groups of collaboration, that agglomerate many publications (for which we labelled a main contributor),
contrast with the large number of groups that have few of them. We identify two main groups: (1) a cohesive one where we labelled the author “Telea, A.”, and (2) another less cohesive but larger one, where we labelled the author “Lanza, M.”. We also observe that red and blue papers agglomerate in the upper and lower part of the visualisation respectively. Although there is no data encoded in the position of rectangles (they are distributed using a force-directed layout), the visualisation facilitates the observation that in small groups only one colour predominates, thus their publications are not intermingled between SOFTVIS and VISSOFT. Moreover, the selected papers are scattered among groups of different size, venues and years of publication. An interactive version of this visualisation is available [24].

C. Data Extraction

Table I presents the attributes that we extracted from each paper: 1) task; 2) need; 3) audience; 4) data source; 5) representation; 6) medium; and 7) tool.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>why the visualisation is needed (e.g., testing)</td>
</tr>
<tr>
<td>Need</td>
<td>which questions motivated the visualisation</td>
</tr>
<tr>
<td>Audience</td>
<td>who will use the visualisation (e.g., analyst)</td>
</tr>
<tr>
<td>Data source</td>
<td>what source of data is visualised (e.g., source code)</td>
</tr>
<tr>
<td>Representation</td>
<td>the data (e.g., pixel-oriented )</td>
</tr>
<tr>
<td>Medium</td>
<td>where to render the visualisation (e.g., wall-disp.)</td>
</tr>
<tr>
<td>Tool</td>
<td>which tool is used for evaluation (e.g., lviz)</td>
</tr>
</tbody>
</table>

We scanned the papers and identified recurrent sections that are likely to contain the data we sought. In our experience, attributes such as task, need, audience and data source are frequently described in the evaluation section, while the representation, medium and tool are typically found in another section dedicated to describe the architectural decisions and implementation of the prototype. Consequently, we extracted the task by identifying frequent terms used to describe development concerns such as programming, testing, debugging, maintenance, reverse-engineering. For the need we looked for questions that are used to specify what can be answered with the visualisation. When there were no explicit questions, we extracted the goal that motivated the need for a proposed visualisation. The audience was detected by identifying roles that users play in development such as programmer, engineer, tester. We extracted the data source by identifying the origin of the software artefacts that are visualised, such as source code and running system. For the representation we reflected on the description of visualisation techniques, analysed figures and looked for their description. We extracted the medium by recognising in the description the technology required to display the visualisation such as wall display, standard monitor. We also extracted attributes of tools from the description of the artefact used in the evaluation such as tool name, and availability. When we were not able to identify an attribute, we searched for common terms already found in other studies. When we still did not find a description, we reported it as not identified.

III. RESULTS

In this section we describe various characteristics of the 65 papers listed in Table II. A complete set of extracted data in our study is available online [24].

A. Task

Table III shows the classification of the papers based on the type of tasks [7] they tackled. Figure 5 shows the distribution of the types of tasks presented in each edition of the venues. We sorted the venues chronologically starting by SOFTVIS editions followed by VISSOFT ones. We think it provides a better understanding of their various contribution. We observe that even though we selected papers from almost all editions of SOFTVIS and VISSOFT (we did not find design study papers in VISSOFT’02), we included only few papers from the first editions of VISSOFT. This can be a consequence of the lower percentage of design study papers in VISSOFT than in SOFTVIS (see Figure 1). We also detected that papers tackling testing appear for the first time only in the two last editions of SOFTVIS and then reappear in VISSOFT’14. An explanation for this can be that those contributors of SOFTVIS interested in visualisation for testing joined VISSOFT once the venues merged. The same explanation we found for papers devoted to visualisation for maintenance tasks that were historically present in SOFTVIS, and that appear in VISSOFT only when the venues fused. Although most of the reviewed studies tackled programming tasks (as shown in Table III) they concentrate on SOFTVIS’03 and VISSOFT’15, showing little presence in the rest of the editions. We realise that the
result provides a good overview of the degree of attention that each development concern has had, but since many different visualisation techniques are proposed within each type, it provides little help to practitioners to find a suitable visualisation for their specific needs.

B. Need

In Table VII we present the developer needs that we identified from studies. Although some studies tackle more than one need we report the most representative one (the complete set of needs is available online†). On the one hand, we found that 75% of studies (i.e., 49) describe envisioned user needs by explicitly posing questions that can be answered using the proposed visualisation, such as “what the software is doing when performance issues arise?” [S49], “what does this called method do?” [S56]. On the other hand, in 25% of studies (i.e., 16) there was no explicit question formulation. In such cases, we identified the goals that the proposed visualisation achieve, examples of them being “to assist designers of scheduling-based, multi-threaded, out-of-core algorithms” [S40], “to get a better insight into the control or data flow inside a program” [S1]. Although questions allow users to assess whether a visualisation is useful, we realise that uncategorised questions hinder the reuse of visualisation. We tackle this issue with a classification of needs based on problem domains. A detailed analysis is provided in Section IV.

C. Audience

Software developers play specific roles such as interaction designer, solution architect, GUI designer, requirements analyst, release coordinator. In contrast, as shown in Table IV, 63% of the studies (i.e., 41) envisioned a generic audience described as maintainer, programmer, developer, engineer,
and user. In the remaining studies the role of the user was more specific such as project manager (6), architect (5), designer (3), or tester (2). Less frequent roles were operation staff, project leader, technical lead, and quality assurance engineer. Some studies envisioned roles of users from other fields such as business owner and student. One study envisioned managers as well as developers pursuing the same questions “(1) when were the changes made? (2) what kind of changes have been made? and (3) how does visit / download time vary over time?” [S17]. Another study envisioned that their tool would be suitable for “everyone involved in software development” [S42]. We realised that a better understanding of the scope of the role that an audience plays would facilitate adoption of visualisation by practitioners.

D. Data source

Table V presents various sources of data that are visualised in the studied papers. The most frequent data were gathered from running system, source code, and version control system. Less frequently, we found non-traditional sources such as documentation, IDE changes, and spreadsheets. Regarding visualisation of source code, the most frequent language supported was Java, followed by C/C++, which was supported by half of the studies. Other languages with little support include Smalltalk and Pascal.

We realise that visualisations have focused on sources of complex data that are difficult to analyse by other means, but this also shows that sources of complex data are not limited to the traditional ones. We also noticed that studies focus mainly on how they modelled data rather than specifying the source and type of data. For instance,
users who are aware of a technique for visualising a stack trace gathered from a running system can decide whether their context is similar enough to adopt the visualisation.

Table V

<table>
<thead>
<tr>
<th>Data source</th>
<th>Reference</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changes</td>
<td>S6, S16</td>
<td>2</td>
</tr>
<tr>
<td>Documentation</td>
<td>S4, S7, S17, S51, S63</td>
<td>5</td>
</tr>
<tr>
<td>Source Code</td>
<td>S1, S3, S8, S9, S11, S12, S14, S15, S16, S18, S22, S24, S26, S27, S31, S32, S35, S36, S57, S58, S64, S65</td>
<td>22</td>
</tr>
<tr>
<td>Spreadsheets</td>
<td>S2</td>
<td>1</td>
</tr>
<tr>
<td>Version Control</td>
<td>S5, S13, S21, S25, S35, S36, S37, S42, S47, S48, S52, S61, S62</td>
<td>14</td>
</tr>
</tbody>
</table>

E. Representation

Describing the representation used in a visualisation is a complex task. Authors proposing a visualisation use various strategies to describe the applied techniques. Some used verbose descriptions ([S42, S45]) by specifying dimensions, metaphors, marks, and properties of them. Others ([S48, S51]) opted for concise but sometimes vague descriptions. We classify the visualisation techniques used in the studies according to the popular taxonomy proposed by Keim [25].

Table VI presents these categories. We notice that almost half of the studies (i.e., 30) combine techniques from several categories. The two most frequent types are Geometrically-Transformed and Dense Pixel. The former type is frequent because node-link techniques that belong to this category are profusely used by visualisations that explore relationships. The latter type contains techniques suitable for depicting massive data sets.

F. Tool

Table VII summarises the tools collected from the papers. Normally, they are developed as prototypes to evaluate a proposed visualisation. Almost all studies (i.e., 98%) introduced a new visualisation tool, but few (i.e., 38%) made their tool and source code publicly available. The exception was a tool named jive [S50, S58]. As one can expect, few prototypes were maintained and extended over time. The most notable cases are teaching tools such as jGrasp and PlanAni. If we consider tools for which current information is available, their average lifespan is 3.7 years. We acknowledge that this value represents only a lower bound, since it does not consider possible earlier presentations of the tools. Various studies often used different visualisation frameworks. OpenGL was the most frequent one used by eight studies over multiple years. Also, four studies used Java3D in more than a decade ago. GraphViz was used by two studies, and more recently, D3 and Roassal were used in some studies. Twenty other studies used multiple frameworks, and in thirty-one there were not explicit information.

G. Medium

When Maletic et al. [7] proposed their taxonomy, they expected that in the future a variety of media would be used by visualisation techniques, however we found few studies exploiting this dimension, shown in Table VIII. Almost 80% of the reviewed studies do not mention the expected medium on which the visualisation should be displayed (labelled as not identified). Among the 20% that explicitly mentioned a medium the majority specified the standard PC display. However, there were others that indicated diverse media from a small window in a standard monitor to a wall-display, large multi-touch tables, and a 3D immersive environment.
**IV. DISCUSSION**

In this section we discuss our findings, and we provide recommendations to practitioners and researchers, respectively, for adopting visualisations, and for identifying domains that require more attention.

A majority of studies do not follow a specific structure for describing their proposed techniques. We believe that following a specific structure (e.g., [7], [9]) encourages researchers to reflect on important dimensions that should drive the design of a visualisation tool. Moreover, we believe that providing a clear description of a research problem, and formulating explicit research questions ease tool adoption by practitioners. For instance, instead of a fuzzy description like “provides an analysis of Java programs” ([864]) which does not reflect an exact goal, we suggest a reformulation to “analyse class dependence for validation of experimental software visualisation techniques”.

**A. RQ1. What are the characteristics of visualisation techniques that support developer needs?**

In section III-A we classified the papers into six high-level software development tasks (shown in Table III). We note
that different visualisation is proposed to tackle developer needs that are classified in the same task. Hence, we argued that such a classification does not provide an appropriate support for practitioners to find and adopt a suitable visualisation for their specific needs. We realise that practitioners require a more fine-grained classification that links existing visualisation techniques to their concrete needs.

We propose to classify the papers into multiple problem domains based on various types of questions that developers ask during software development [16]–[18]. Such questions reflect developer needs and we believe mapping them to existing visualisation techniques provides a better support for practitioners to adopt a visualisation in their daily tasks. We applied the classification proposed by LaToza et al. [48]. It comprises 21 problem domains which they used to categorise 94 types of questions. According to our investigation, this classification offers an appropriate granularity to accommodate the questions from other studies too. Hence, we classified the 65 included papers by identifying problem domains that contain similar types of questions to the needs extracted from the papers (shown in Table VII). In studies which we extracted a goal instead of a question, we inferred the problem domain from other types of questions that would help users to achieve that goal. Table IX presents the obtained results.

While few problem domains in the classification (like debugging and testing) seem to be a task by themselves, they also occur very often in the context of addressing different tasks. That is, a visualisation proposed to support questions regarding performance during a maintenance task (e.g., “where is most of the time being spent?” [S44]) may differ from the one proposed for performance questions that arise during a debugging session (e.g., “what are these event pair sequences?” [S34]). Figure 6 shows the mapping between the problem domains and the types of visualisation techniques. In it, problem domains are labelled. The ones in the same category are vertically aligned (left-to-right changes, element relationships, and elements). The colours of the tiles encode the type of visualisation technique used by studies tackling that domain. Problem domains that did not match any studies are shown in black. The size of a tile is proportional to the number of studies classified in that domain. Looking at the distribution of visualisation techniques across the types of problem domains (i.e., changes, element relationships and elements) we do not perceive a preferred one. Instead, we observe that dense pixel and geometrically-transformed are the most frequent techniques used in the main problem domains such as history, debugging, performance. In contrast, iconic techniques are present in only a few domains, but when present they predominate over other techniques such as history, implications and testing. Iconic techniques enforce comparison of multivariate data by mapping their properties to the various dimensions of a glyph (including its position). Questions regarding the history domain frequently involve the time which is commonly mapped to the position. We think that this is the reason why most visualisations proposed to tackle needs in the history domain include iconic techniques.

B. RQ2. How well are various problem domains supported by visualisation?

We estimate the importance of a problem domain for practitioners based upon a previous study conducted by LaToza et al. [48]. The more types of questions a problem domain contains, the more important that domain is for developers. The double bar-chart in Figure 7 compares the importance of developer needs (red axis on top) versus the number of visualisation techniques that address these needs (grey axis at bottom). Problem domains (at left) are coloured to encode the category that they belong to (changes in green, element relationships in red, and elements in blue), and are sorted decreasingly from the most important one for practitioners.

We learned that practitioners are more concerned about changes, while existing visualisations distribute their attentions among all three categories. Some problem domains (e.g., rationale, intent, implementation, and refactoring) are very important for developers but have little visualisation support. In contrast, several less important problem domains (e.g., history, performance, concurrency and dependencies) received a good degree of attention. We wonder why some are not supported? We conjecture that less well-supported domains tackle problems that require
hidden semantics to be inferred from software artefacts, so proposing a visualisation is difficult.

C. Threats to Validity

The main threat to the validity of our study is bias in paper selection. We did not include papers from other venues. We mitigated this threat by selecting peer-reviewed papers from the most cited venues that dedicate to software visualisation. Moreover, we included design studies and excluded other types of papers. However, since most of papers do not specify their types, we may have missed some. We mitigated this threat by defining a cross-checking procedure and criteria for paper type classification. Finally, the data extraction process could be biased. We mitigated this by establishing a protocol to extract the data of each paper equally; and by maintaining a spreadsheet to keep records, normalise terms, and identify anomalies.

V. CONCLUSION

In this paper we studied 65 publications in academia that describe how visualisation techniques can help developers to carry out their tasks, and we investigated how well practitioner needs are supported by existing visualisation techniques. On the one hand, we analysed research that describes complex questions that practitioners often ask during software development. On the other hand, we reviewed the literature looking for the needs that benefit from particular visualisations. We compared the degree of importance of need in various problem domains for practitioners to the visualisation support available for those domains. We found a disconnect between the problem domains on which visualisation have focused and the domains that get the most attention from practitioners. We realised some problem domains such as rationale, intent and implementation, refactoring, implementing, contracts, and policies require more attention from the visualisation community; while a good amount of work devoted to history, performance, concurrency and dependencies.

This paper makes the following contributions:

• A study of the characteristics of existing research in the field of software visualisation.
• An analysis of the relation between practitioner needs and current visualisation techniques.

We observe that as researchers in the field we lack a method to delimit the art and science inherently involved in developing visualisation tools and techniques. We need a systematic way to develop software visualisations that eases their adoption by practitioners. Consequently, we plan to expand this work by proposing such a method.

ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project “Agile Software Analysis” (SNSF project No. 200020-162352, Jan 1, 2016 - Dec. 30, 2018). Merino has been partially funded by CONICYT BCH/Doctorado Extranjero 72140330.