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.

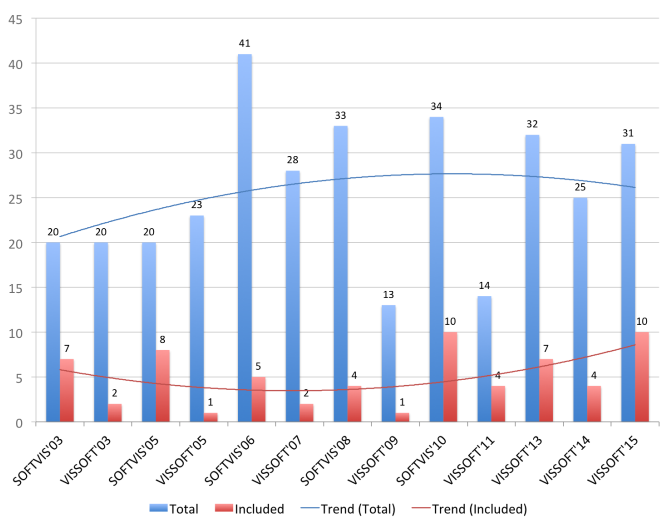

Figure 1. Collection of 346 papers published in SOFTVIS/VISSOFT venues.

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.

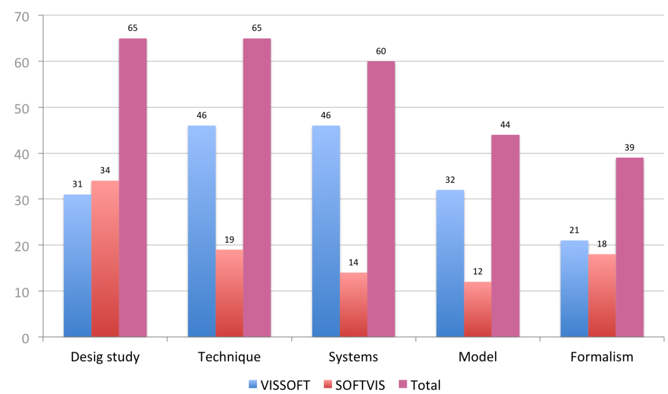

Figure 2. Classification of the 273 SOFTVIS/VISSOFT papers by type.

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 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),

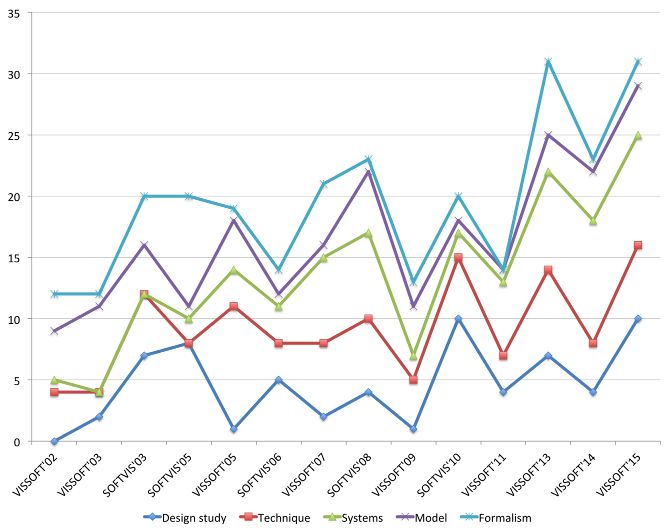

Figure 3. Evolution of SOFTVIS/VISSOFT papers by type. From the bottom upwards: *Design Study*, *Technique*, *Systems*, *Model*, and *Formalism*.

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 I
DATA EXTRACTED FROM PAPERS.

Attribute	**Description**
Task	*why* the visualisation is needed (*e.g.*, testing)
Need	*which* questions motivated the visualisation
Audience	*who* will use the visualisation (*e.g.*, analyst)
Data source	*what* source of data is visualised (*e.g.*, source code)
Representation	*what* technique is used to represent the data (*e.g.*, pixel-oriented)
Medium	*where* to render the visualisation (*e.g.*, wall-disp.)
Tool	*which* tool is used for evaluation (*e.g.*, lviz)

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

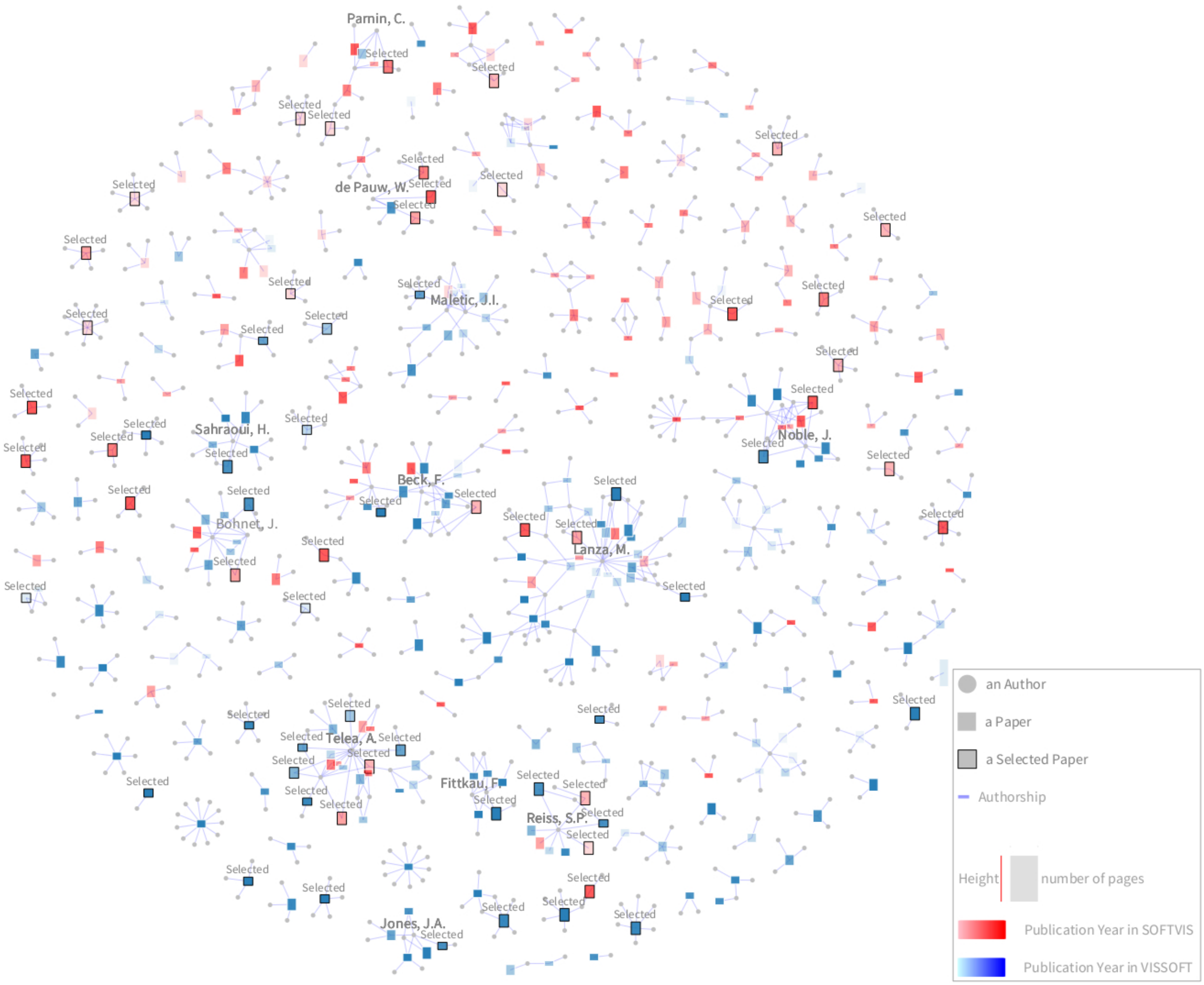

Figure 4. Overview of the complete publication record of SOFTVIS/VISSOFT. The 65 selected papers (out of 346) correspond to design studies.

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,*

Table II
THE 65 INCLUDED PAPERS IN THE STUDY.

Id	Reference	Year
[S1]	Kayrebt: An Activity Diagram Extraction and Visualization Toolset Designed for the Linux Codebase, **Georget, L. *et al.***	2015
[S2]	XVIZIT: Visualizing Cognitive Units in Spreadsheets, **Hodnigg, K. *et al.***	2015
[S3]	Vestige: A Visualization Framework for Engineering Geometry-Related Software, **Schneider, T. *et al.***	2015
[S4]	Hierarchical Software Landscape Visualization for System Comprehension: A Controlled Experiment, **Fittkau, F. *et al.***	2015
[S5]	Interactive Tag Cloud Visualization of Software Version Control Repositories, **Greene, G.J. *et al.***	2015
[S6]	Blended, Not Stirred: Multi-concern Visualization of Large Software Systems, **Dal Sasso, T. *et al.***	2015
[S7]	Pixel-Oriented Techniques for Visualizing Next-Generation HPC Systems, **Cottam, J. *et al.***	2015
[S8]	SMNLV: A Small-Multiples Node-Link Visualization Supporting Software Comprehension by Displaying Multiple Relationships in Software Structure, **Abuthawabeh, A. *et al.***	2015
[S9]	Live Visualization of GUI Application Code Coverage with GUITracer, **Molnar, A.J. *et al.***	2015
[S10]	Advancing Data Race Investigation and Classification through Visualization, **Koutsopoulos, N. *et al.***	2015
[S11]	Visual Clone Analysis with SolidSDD, **Voinea, L. *et al.***	2014
[S12]	Polyptychon: A Hierarchically-Constrained Classified Dependencies Visualization, **Daniel, D.T. *et al.***	2014
[S13]	ChronoTwigger: A Visual Analytics Tool for Understanding Source and Test Co-evolution, **Ens, B. *et al.***	2014
[S14]	Visualizing the Evolution of Systems and Their Library Dependencies, **Kula, R.G. *et al.***	2014
[S15]	The visualizations of code bubbles, **Reiss, S.P. *et al.***	2013
[S16]	Visualizing software dynamicities with heat maps , **Benomar, O. *et al.***	2013
[S17]	DEVis: A tool for visualizing software document evolution, **Junji Zhi *et al.***	2013
[S18]	SourceVis: Collaborative software visualization for co-located environments , **Anslow, C. *et al.***	2013
[S19]	SYNCTRACE: Visual thread-interplay analysis, **Karran, B. *et al.***	2013
[S20]	Automatic categorization and visualization of lock behavior, **Reiss, S.P. *et al.***	2013
[S21]	Chronos: Visualizing slices of source-code history, **Servant, F. *et al.***	2013
[S22]	Visual support for porting large code bases, **Broeksema, B. *et al.***	2011
[S23]	Visualising concurrent programs with dynamic dependence graphs, **Lonnberg, J. *et al.***	2011
[S24]	Visual exploration of program structure, dependencies and metrics with SolidSX, **Reniers, D. *et al.***	2011
[S25]	MosaiCode: Visualizing large scale software: A tool demonstration , **Maletic, J.I. *et al.***	2011
[S26]	An interactive ambient visualization for code smells, **Murphy-Hill, E. *et al.***	2010
[S27]	Exploring the inventor's paradox: applying jigsaw to software visualization, **Ruan, H. *et al.***	2010
[S28]	Towards anomaly comprehension: using structural compression to navigate profiling call-trees, **Lin, S. *et al.***	2010
[S29]	Heapviz: interactive heap visualization for program understanding and debugging, **Aftandilian, E.E. *et al.***	2010
[S30]	Trevis: a context tree visualization & analysis framework and its use for classifying performance failure reports, **Adamoli, A. *et al.***	2010
[S31]	Dependence cluster visualization, **Islam, S.S. *et al.***	2010
[S32]	Embedding spatial software visualization in the IDE: an exploratory study, **Kuhn, A. *et al.***	2010
[S33]	Visualizing windows system traces, **Wu, Y. *et al.***	2010
[S34]	Zinsight: a visual and analytic environment for exploring large event traces, **de Pauw, W. *et al.***	2010
[S35]	Representing development history in software cities, **SteinbrÃijckner, F. *et al.***	2010
[S36]	Case study: Visual analytics in software product assessments, **Telea, A. *et al.***	2009
[S37]	Representing unit test data for large scale software development, **Cottam, J.A. *et al.***	2008
[S38]	A catalogue of lightweight visualizations to support code smell inspection, **Parnin, C. *et al.***	2008
[S39]	Streamsight: a visualization tool for large-scale streaming applications, **de Pauw, W. *et al.***	2008
[S40]	Stacked-widget visualization of scheduling-based algorithms, **Bernardin, T. *et al.***	2008
[S41]	Visualizing Dynamic Memory Allocations, **Moreta, S. *et al.***	2007
[S42]	A Visualization for Software Project Awareness and Evolution , **Ripley, R.M. *et al.***	2007
[S43]	Experimental evaluation of animated-verifying object viewers for Java, **Jain, J. *et al.***	2006
[S44]	Execution patterns for visualizing web services, **de Pauw, W. *et al.***	2006
[S45]	Visualizing live software systems in 3D, **Greevy, O. *et al.***	2006
[S46]	Visual exploration of function call graphs for feature location in complex software systems, **Bohnet, J. *et al.***	2006
[S47]	Multiscale and multivariate visualizations of software evolution, **Voinea, L. *et al.***	2006
[S48]	CVSscan: visualization of code evolution, **Voinea, L. *et al.***	2005
[S49]	Jove: java as it happens, **Reiss, S.P. *et al.***	2005
[S50]	Methodology and architecture of JIVE, **Gestwicki, P. *et al.***	2005
[S51]	Visual Exploration of Combined Architectural and Metric Information, **Termeer, M. *et al.***	2005
[S52]	Visual data mining in software archives, **Burch, M. *et al.***	2005
[S53]	The war room command console: shared visualizations for inclusive team coordination, **O'Reilly, C. *et al.***	2005
[S54]	Visualizing structural properties of irregular parallel computations, **Blochinger, W. *et al.***	2005
[S55]	Visualization of mobile object environments, **Frishman, Y. *et al.***	2005
[S56]	Towards understanding programs through wear-based filtering, **DeLine, R. *et al.***	2005
[S57]	Program animation based on the roles of variables, **Sajaniemi, J. *et al.***	2003
[S58]	Visualizing Java in action, **Reiss, S.P. *et al.***	2003
[S59]	EVolve: an open extensible software visualization framework, **Wang, Q. *et al.***	2003
[S60]	Visualization of program-execution data for deployed software, **Orso, A. *et al.***	2003
[S61]	A system for graph-based visualization of the evolution of software, **Collberg, C. *et al.***	2003
[S62]	Interactive locality optimization on NUMA architectures, **Mu, T. *et al.***	2003
[S63]	Graph visualization for the analysis of the structure and dynamics of extreme-scale supercomputers, **Zhou, C. *et al.***	2003
[S64]	KScope: A Modularized Tool for 3D Visualization of Object-Oriented Programs, **Davis, T.A. *et al.***	2003
[S65]	Self-Organizing Maps Applied in Visualising Large Software Collections, **Brittle, J. *et al.***	2003

Table III
CLASSIFICATION OF PAPERS BASED ON THE TASKS.

Task	Reference	#
Debugging	S3, S10, S15, S29, S34, S46, S49, S50, S55, S58, S60	11
Maintenance	S11, S14, S16, S18, S19, S22, S25, S26, S27, S31, S38, S44, S48	13
Programming	S2, S5, S7, S9, S20, S23, S32, S33, S40, S41, S43, S51, S54, S56, S57, S59, S61, S62, S63	19
Reverse Engineering	S1, S6, S8, S12, S21, S24, S28, S35, S45, S47, S52, S64, S65	13
Software Process Management	S4, S17, S36, S42, S53	5
Testing	S13, S30, S32, S37, S39	5

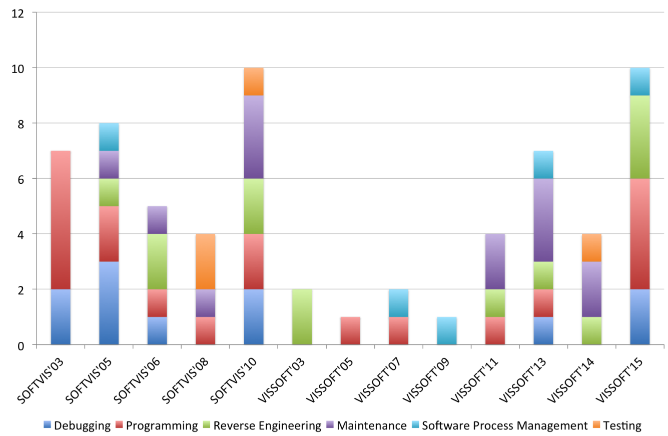

Figure 5. Distribution of papers by task in each venue.

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 describing how they modelled data rather than specifying the source and type of data. For instance,

Table IV
CLASSIFICATION OF PAPERS BASED ON THE AUDIENCE.

Audience	Reference	#
Architect	S12, S25, S36, S48, S51	5
Business Owner	S44	1
Designer	S26, S40, S44	3
Developer	S3, S6, S7, S11, S12, S16, S17, S18, S21, S22, S24, S27, S28, S30, S31, S32, S35, S37, S38, S39, S44, S45, S46, S47, S48, S51, S53, S55, S56, S60	30
Engineer	S31, S45, S50, S52	4
Everyone	S42	1d
Maintainer	S14, S31, S48, S60	4
Manager	S17	1
Operation Staff	S44	1
Performance Analyst	S34	1
Project Manager	S25, S35, S36, S38, S48, S53	6
Practitioner	S9	1
Project Leader	S22	1
Programmer	S20, S26, S29, S44, S49	5
Quality Assurance Engineer	S10	1
Researcher	S64	1
Student	S9, S23, S43, S57	4
Technical Leader	S53	1
Test Manager	S13	1
Tester	S44, S48	1
User	S1, S2, S4, S5, S8, S15, S19, S24, S33, S41, S54, S58, S59, S61, S62, S63, S65	17

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
CLASSIFICATION OF PAPERS BASED ON THE DATA SOURCE.

Data source	Reference	#
Changes	S6, S16	2
Documentation	S4, S7, S17, S51, S63	5
Running System	S10, S19, S20, S23, S28, S29, S30, S33, S34, S38, S39, S40, S41, S43, S44, S45, S46, S49, S50, S54, S55, S59, S60	23
Source Code	S1, S3, S8, S9, S11, S12, S14, S15, S16, S18, S22, S24, S26, S27, S31, S32, S53, S56, S57, S58, S64, S65	22
Spreadsheets	S2	1
Version Control System	S5, S13, S21, S25, S35, S36, S37, S42, S47, S48, S52, S61, S62	14

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.

Table VI
CLASSIFICATION OF PAPERS BASED ON THE REPRESENTATION.

Representation	Reference	#
Dense Pixel	S7, S11, S15, S16, S19, S20, S22, S24, S25, S31, S33, S34, S36, S38, S40, S41, S47, S48, S52, S53, S60, S62	22
Geometrically-Transformed	S3, S4, S8, S12, S13, S14, S16, S17, S18, S19, S20, S21, S22, S23, S24, S27, S29, S36, S37, S40, S44, S45, S49, S52, S54, S61	26
Iconic	S5, S6, S14, S17, S26, S27, S28, S32, S37, S42, S49, S51, S63, S65	14
Stacked	S1, S6, S10, S11, S16, S18, S24, S30, S35, S36, S49, S58, S60, S63, S64	15
Standard 2D/3D	S2, S9, S39, S43, S44, S46, S50, S55, S56, S57, S59	11

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.

Table VII
VISUALISATION TOOLS AND NEEDS INTRODUCED BY PAPERS.

Ref.	Tool	Year	Framework	Technology	Questions and Goals that Motivate Visualisation
[S1]	Kayrebt [26]	2015	Extractor, Globsym, Viewer	C	to get a better insight of the control or data flow inside a program
[S2]	XVIZIT [27]	2015	Java FX, Control FX, GraphStream	Excel and Numbers spreadsheet	are there modules or self-contained computations?
[S3]	Vestige	2015	C++, OpenGL	-	how the computation reached that result?
[S4]	ExploreViz	2015	-	-	which applications are duplicated on multiple nodes?
[S5]	ConceptCloud [28]	2015	-	SVN, Git	which developers collaborate?
[S6]	Blended City	2015	Roassal	Smalltalk	what happened to our system recently?
[S7]	Vampir	2015	-	HPX C++	how different are work queues on different threads?
[S8]	SMNLV	2015	Java 8, Graphisto, abego, NetBeans Visual Library.	Java	to check guidelines and re-engineering of existing software
[S9]	GUITracer [29]	2015	Java 6	Java using AWT, Swing, SWT	how the GUI and the underlying code are related?
[S10]	RaceView	2015	Eclipse Visualization Toolkit (Zest)	C	how a specific code location can be reached via function calls?
[S11]	SolidSDD [30]	2014	C++, OpenGL	C/C++, Java, C#	how are clones distributed in system structure?
[S12]	Polyptychon	2014	D3	Java	how the system is actually organized?
[S13]	ChronoTwigger	2014	OpenGL, GLUT, VR Juggler	Git	what test files changed compared to source files at the beginning of a project?
[S14]	-	2014	-	Java	how the dependency relation between a system and its dependencies evolves?
[S15]	Code Bubbles [31]	2013	-	Java	what other programmers are working on?
[S16]	VERSO	2013	-	Java	what are coworkers working on?
[S17]	DEVis [32]	2013	G4P	-	what kind of changes have been made?
[S18]	SourceVis	2013	MT4j, OpenCloud, JFreeChart		how many versions contain annotation classes?
[S19]	SYNCTRACE	2013	-	-	where and when a thread waits or releases?
[S20]	-	2013	-	Java	how much time is spent blocking on a specific lock?
[S21]	Chronos	2013	Java	CVS, SVN, Git	when, how, by whom, and why was this code changed or inserted?
[S22]	KDevelop [33]	2011	C++	C/C++	which code fragments are affected by a given rewrite rule?
[S23]	Atropos [34]	2011	-	-	how the operations performed during the execution?
[S24]	SolidSX [30]	2011	OpenGL, GLUT, FTGL, wxWidgets	.NET/VB/C#, Java, Visual C++	to explore large compound attributed graph for program comprehension
[S25]	MosaiCode	2011	C++, Qt	-	what are areas of the code with high code churn?
[S26]	Stench Blossom [35]	2010	-	Java	how widespread the feature envy is?
[S27]	Jigsaw [36]	2010	-	-	what other entities are likely to de- pend on this package?
[S28]	ProfVis [37]	2010	-	Java	to diagnose anomalies in programs based on partial knowledge
[S29]	Heapviz	2010	Prefuse toolkit	Java	how the program works and why it doesn't work?
[S30]	Trevis	2010	Trevis, GraphViz	Java	where the program spent most time and which methods were called?
[S31]	Decluvi	2010	-	-	helps to quickly identify computations involved in clusters of dependence
[S32]	CodeMap [38]	2010	-	Java	which architectural paradigm is used?
[S33]	lviz	2010	OpenJDK 1.6.0 18 (64bit)	VDP	how the operating system works to examine performance problems?
[S34]	Zinsight [39]	2010	-	IBM System z	how did we get to these events?
[S35]	CrocoCosmos	2010	jMonkeyEngine	Java	what is the history of a piece of code?
[S36]	-	2009	-	Keil C166	why it was hard to add new features to the existing software?
[S37]	SeeTest	2008	Stencil visualization environment	-	how did the changes from yesterday effect the stability of the project?
[S38]	NosePrints	2008	-	-	where the value was originating from?
[S39]	Streamsight	2008	dot	IBM System S	what kind of hardware interconnect link technology to employ?
[S40]	Lumiere	2008	-	-	to assist designers of scheduling-based multi-threaded algorithms
[S41]	MemoView	2007	-	-	how does the allocator speed depend on type and parameters?
[S42]	Paladir	2007	Palantir	Java	when was the artifact changed?
[S43]	jGrasp [40]	2006	-	Java	to understand concepts of dynamic programming implementation
[S44]	IBM WS Navigator	2006	-	-	where most of the time is being spent?
[S45]	TraceCrawler	2006	CCJun	Smalltalk	which parts of the code are active during the execution of a feature?
[S46]	Call Graph Analyzer	2006	GraphViz	C/C++	which the important functions for feature understanding are?
[S47]	CVSgrab	2006	-	-	what versions, with these words in the log, were committed by that her?
[S48]	CVSscan [41]	2005	-	-	which parts of the code are unstable?
[S49]	Jove [42]	2005	-	Java	what the software is doing when performance issues arise?
[S50]	Jive [43]	2005	-	Java	runtime comprehension of object-oriented programs
[S51]	MetricView [44]	2005	C++, OpenGL, FreeType, wxWindows	-	what happens If I change this component?
[S52]	EPOSee [45]	2005	-	-	what items have been changed at the same time?
[S53]	War Room Console	2005	Java, C++	Java, C/C++	which artifacts are of importance?
[S54]	DOT	2005	Java, yFiles library	C/C++	to identify optimal parameters to distribute the work on the processors
[S55]	Mobile Object Vis.	2005	Java3D	-	to understand connections in the distributed object network
[S56]	FAN	2005	-	-	what does this called method do?
[S57]	PlanAni [46]	2003	Java3D	Pascal	what part of the array is currently being sorted?
[S58]	Jive [43]	2003	-	Java	how much time each thread spent in each class?
[S59]	EVolve [47]	2003	-	Java	help us to develop compiler optimization
[S60]	Gamma/Gammatella	2003	Java, Swing, TreeMap Java Library	Java	investigate the behavior of deployed software
[S61]	GEVOL	2003	-	-	who was responsible for which parts of the program during which periods?
[S62]	-	2003	-	-	helps users to allocate data onto their dominating nodes
[S63]	Flatland	2003	OpenGL	-	analysis of massively parallel supercomputer architectures
[S64]	Kscope	2003	Java3D	Java	to provide an analysis of java programs
[S65]	GENISOM	2003	Java3D	-	to aid programmers in the process of reverse engineering

Table VIII
CLASSIFICATION OF PAPERS BASED ON THE MEDIUM.

Medium	Reference	#
Immersive 3D Environment	S13	1
Muti-Touch Tables	S18	1
Not Identified	S1, S2, S3, S5, S6, S7, S9, S10, S11, S12, S14, S15, S16, S17, S19, S20, S21, S22, S23, S25, S26, S27, S28, S29, S30, S32, S33, S34, S35, S36, S37, S39, S40, S41, S43, S44, S46, S47, S49, S50, S52, S54, S55, S56, S57, S58, S59, S60, S61, S62, S65	51
Standard Screen	S4, S8, S24, S31, S38, S42, S45, S48, S51, S63, S64	11
Wall Display	S8, S38, S42, S53	4

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*" (*[S64]*) which does not reflect an exact goal, we suggest a reformulation to "*analyse class dependency 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

Table IX
CLASSIFICATION OF PAPERS BASED ON THE NEEDS.

Problem Domain		Reference	#
Changes	Building and branching	-	0
	Debugging	S3, S15, S23, S29, S32, S34, S39, S46, S49, S50, S58, S60	12
	History	S5, S6, S16, S17, S21, S35, S37, S42, S47, S48, S52	11
	Implementing	S3, S39, S43	3
	Implications	S14, S22, S26, S28, S51, S59	6
	Policies	-	0
	Rationale	S61	1
	Refactoring	-	0
	Teammates	S16, S18	2
	Testing	S3, S32, S38, S64	4
Element relationship	Architecture	S4, S11, S13, S25, S32, S36, S63, S65	8
	Contracts	S8	1
	Control flow	S1, S45	2
	Data flow	S1	1
	Dependencies	S2, S8, S12, S24, S27, S31, S55, S56	8
	Type relationships	-	0
Elements	Concurrency	S7, S10, S19, S20, S23, S40	6
	Intent and implication	S32, S53, S56	3
	Location	S9, S32, S45, S57	4
	Method properties	-	0
	Performance	S30, S33, S34, S39, S41, S44, S54, S62	8

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

Figure 6. Mapping type of visualisation used by studies to problem domains.

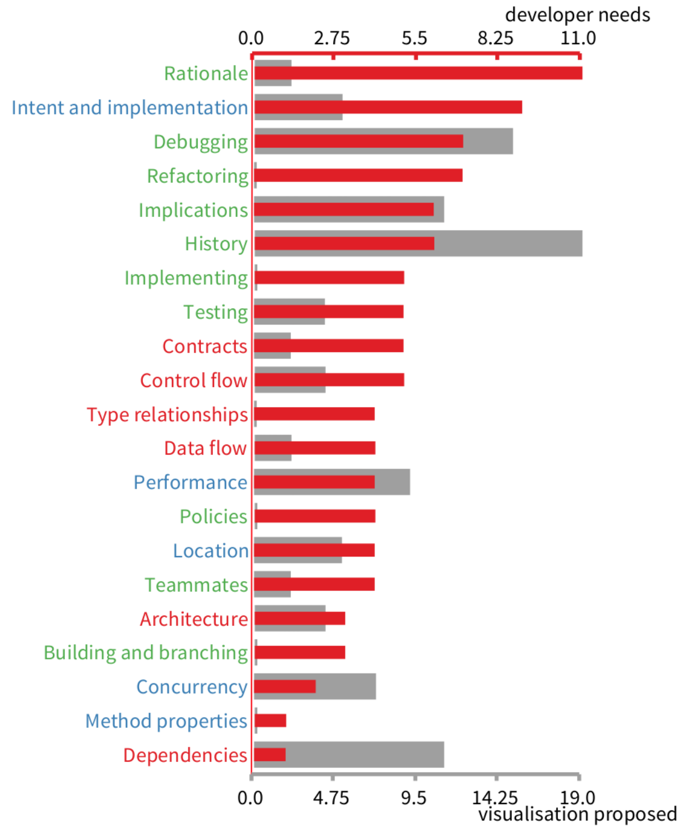

Figure 7. Comparing the degree of importance of developer needs vs. their visualisation support by problem domain.

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.

REFERENCES

[1] R. Theron, A. Gonzalez, and F. J. Garcia, "Supporting the understanding of the evolution of software items," in *Proceedings of the 4th ACM symposium on Software visualization*. ACM, 2008, pp. 189–192.

[2] W. De Pauw, S. Krasikov, and J. Morar, "Execution patterns for visualizing web services," in *Proceedings ACM International Conference on Software Visualization (SoftVis'06)*. New York NY: ACM Press, Sep. 2006.

[3] W. De Pauw and S. Heisig, "Zinsight: a visual and analytic environment for exploring large event traces," in *Proceedings of the 5th international symposium on Software visualization*, ser. SOFTVIS '10. New York, NY, USA: ACM, 2010, pp. 143–152. [Online]. Available: http://doi.acm.org/10.1145/1879211.1879233

[4] Y. Park and C. Jensen, "Beyond pretty pictures: Examining the benefits of code visualization for open source newcomers," in *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*. IEEE, 2009, pp. 3–10.

[5] N. Faltin, "Structure and constraints in interactive exploratory algorithm learning," in *Software Visualization*. Springer, 2002, pp. 213–226.

[6] S. P. Reiss, "JOVE: Java as it happens," in *Proceedings of SoftVis 2005(ACM Symposium on Software Visualization)*, 2005, pp. 115–124.

[7] J. I. Maletic, A. Marcus, and M. Collard, "A task oriented view of software visualization," in *Proceedings of the 1st Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2002)*. IEEE, Jun. 2002, pp. 32–40.

[8] M. Schots and C. Werner, "Using a task-oriented framework to characterize visualization approaches," in *Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on*. IEEE, 2014, pp. 70–74.

[9] M.-A. D. Storey, D. Čubranić, and D. M. German, "On the use of visualization to support awareness of human activities in software development: a survey and a framework," in *SoftVis'05: Proceedings of the 2005 ACM symposium on software visualization*. ACM Press, 2005, pp. 193–202. [Online]. Available: http://portal.acm.org/citation.cfm?id=1056018.1056045

[10] H. M. Kienle and H. A. Muller, "Requirements of software visualization tools: A literature survey," *VISSOFT 2007. 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pp. 2–9, 2007.

[11] H. Padda, A. Seffah, and S. Mudur, "Visualization patterns: A context-sensitive tool to evaluate visualization techniques," in *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*. IEEE, 2007, pp. 88–91.

[12] M. Sensalire, P. Ogao, and A. Telea, "Classifying desirable features of software visualization tools for corrective maintenance," in *Proceedings of the 4th ACM symposium on Software visualization*. ACM, 2008, pp. 87–90.

[13] K. Gallagher, A. Hatch, and M. Munro, "A framework for software architecture visualization assessment," in *VISSOFT*. IEEE CS, Sep. 2005, pp. 76–81.

[14] J. Paredes, C. Anslow, and F. Maurer, "Information visualization for agile software development," in *Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on*. IEEE, 2014, pp. 157–166.

[15] M. Shahin, P. Liang, and M. A. Babar, "A systematic review of software architecture visualization techniques," *Journal of Systems and Software*, vol. 94, pp. 161–185, 2014.

[16] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, ser. SIGSOFT '06/FSE-14. New York, NY, USA: ACM, 2006, pp. 23–34. [Online]. Available: http://people.cs.ubc.ca/~murphy/papers/other/asking-answering-fse06.pdf

[17] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proceedings of the 29th international conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 344–353.

[18] T. Fritz and G. C. Murphy, "Using information fragments to answer the questions developers ask," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 175–184. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806828

[19] S. Keele, "Guidelines for performing systematic literature reviews in software engineering," Technical report, EBSE Technical Report EBSE-2007-01, Tech. Rep., 2007.

[20] (2016) SoftVis. [Online]. Available: http://dl.acm.org/event.cfm?id=RE322

[21] (2016) VISSOFT. [Online]. Available: http://ieeexplore.ieee.org/xpl/conhome.jsp?punumber=1001231

[22] (2016) CORE. [Online]. Available: http://portal.core.edu.au/conf-ranks/

[23] T. Munzner, "Process and pitfalls in writing information visualization research papers," in *Information visualization*. Springer, 2008, pp. 134–153.

[24] L. Merino. (2016) Replication package. towards actionable visualisation in software development. [Online]. Available: http://scg.unibe.ch/research/visualisation-review

[25] D. A. Keim, "Information visualization and visual data mining," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 8, no. 1, pp. 1–8, 2002.

[26] L. Georget. (2015) Kayrebt. [Online]. Available: https://github.com/lgeorget/Kayrebt-Dumper

[27] K. Hodnigg. (2015) XVIZIT. [Online]. Available: https://xvizit.wordpress.com/portfolio/metrics-based-spreadsheet-visualization/

[28] G. Greene. (2015) ConceptCloud. [Online]. Available: http://www.conceptcloud.org

[29] A. Molnar. (2015) GUITracer. [Online]. Available: https://bitbucket.org/guiresearch/tools

[30] L. Voinea. (2014) SolidSDD. [Online]. Available: http://www.solidsourceit.com/index.html

[31] S. Reiss. (2013) Code Bubbles. [Online]. Available: http://cs.brown.edu/~spr/codebubbles/

[32] J. Zhi. (2013) DEVis. [Online]. Available: https://sites.google.com/site/junjizhi/devis_tool

[33] B. Broeksema. (2011) KDevelop. [Online]. Available: http://www.gitorious.org/kdevcpptools/kdevcpptools

[34] J. Lonnberg. (2011) Atropos. [Online]. Available: http://www.cse.hut.fi/en/research/LeTech/Atropos/

[35] E. Murphy-Hill. (2010) Stench Blossom. [Online]. Available: https://github.com/DeveloperLiberationFront/refactoring-tools/tree/master/installables/update_sites/stench_blossom

[36] H. Ruan. (2010) Jigsaw. [Online]. Available: http://www.cc.gatech.edu/gvu/ii/jigsaw/

[37] S. Lin. (2010) Profvis. [Online]. Available: http://ftaiani.ouvaton.org/7-software/profvis.html

[38] A. Kuhn. (2010) CodeMap. [Online]. Available: http://github.com/akuhn/codemap

[39] W. de Pauw. (2010) Zinsight. [Online]. Available: http://researcher.watson.ibm.com/researcher/view_group.php?id=613

[40] J. Jain. (2006) jGrasp. [Online]. Available: http://www.jgrasp.org/

[41] L. Voinea. (2005) CVSscan. [Online]. Available: http://www.win.tue.nl/vis1/home/lvoinea/VCN.html

[42] S. Reiss. (2005) Jove. [Online]. Available: http://cs.brown.edu/people/spr/research/visjove.html

[43] P. Gestwicki. (2005) Jive. [Online]. Available: http://cs.brown.edu/~spr/research/vizjive.html

[44] M. Termeer. (2005) MetricView. [Online]. Available: http://www.win.tue.nl/san/projects/empanada/metricview/

[45] M. Burch. (2005) EPOSee. [Online]. Available: http://www.st.uni-trier.de/eposoft/eposee/

[46] J. Sajaniemi. (2003) PlanAni. [Online]. Available: http://www.cs.uef.fi/~saja/var_roles/planani/index.html

[47] Q. Wang. (2003) EVolve. [Online]. Available: http://www.sable.mcgill.ca/evolve/

[48] T. D. LaToza and B. A. Myers, "Hard-to-answer questions about code," in *Evaluation and Usability of Programming Languages and Tools*, ser. PLATEAU '10. New York, NY, USA: ACM, 2010, pp. 8:1–8:6. [Online]. Available: http://doi.acm.org/10.1145/1937117.1937125