Information and Software Technology 55 (2013) 1860–1883

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

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

Software evolution visualization: A systematic mapping study

Renato Lima Novais [a,d,*], André Torres [a], Thiago Souto Mendes [a,d], Manoel Mendonça [a,b], Nico Zazworka [c]

[a] *Computer Science Department, Federal University of Bahia, Bahia, Brazil*
[b] *Fraunhofer Project Center for Software and Systems Engineering, Bahia, Brazil*
[c] *Fraunhofer Center for Experimental Software Engineering, MD, USA*
[d] *Information Technology Department, Federal Institute of Bahia, Campus Santo Amaro, Bahia, Brazil*

ARTICLE INFO

Article history:
Received 5 October 2012
Received in revised form 24 May 2013
Accepted 31 May 2013
Available online 11 June 2013

Keywords:
Systematic mapping study
Software visualization
Software evolution

ABSTRACT

Background: Software evolution is an important topic in software engineering. It generally deals with large amounts of data, as one must look at whole project histories as opposed to their current snapshot. Software visualization is the field of software engineering that aims to help people to understand software through the use of visual resources. It can be effectively used to analyze and understand the large amount of data produced during software evolution.
Objective: This study investigates Software Evolution Visualization (SEV) approaches, collecting evidence about how SEV research is structured, synthesizing current evidence on the goals of the proposed approaches and identifying key challenges for its use in practice.
Methods: A mapping study was conducted to analyze how the SEV area is structured. Selected primary studies were classified and analyzed with respect to nine research questions.
Results: SEV has been used for many different purposes, especially for change comprehension, change prediction and contribution analysis. The analysis identified gaps in the studies with respect to their goals, strategies and approaches. It also pointed out to a widespread lack of empirical studies in the area.
Conclusion: Researchers have proposed many SEV approaches during the past years, but some have failed to clearly state their goals, tie them back to concrete problems, or formally validate their usefulness. The identified gaps indicate that there still are many opportunities to be explored in the area.

Contents

* Corresponding author at: Computer Science Department, Federal University of Bahia, Bahia, Brazil. Tel.: +55 7181485664.
E-mail addresses: renatoln@dcc.ufba.br (R.L. Novais), atorres@dcc.ufba.br (A. Torres), thiagomendes@dcc.ufba.br (T.S. Mendes), mgmendonca@dcc.ufba.br (M. Mendonça), nzazworka@fc-md.umd.edu (N. Zazworka).

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.05.008

1. Introduction

Software evolution has been highlighted as one of the most important topic in software engineering and maintenance. It generally deals with large amounts of data that originates from heterogeneous sources such as Source Code Management (SCM) repositories, Bug Tracking Systems (BTS), mailing and project discussion lists. One of the key aspects of software evolution is to build 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.

Software Visualization (SoftVis) is the field of Software Engineering (SE) that aims to help people to understand software through the use of visual resources [9], and it can be effectively used to analyze and understand the large amount of data produced during software evolution. For this reason, many researchers have been proposing software evolution visualization (SEV) tools ([S9, S26, S47, S101, S115, S127]).[1] Generally, these tools aim to analyze the evolution of the software with respect to a set of software maintenance related questions.

The body of work on SEV is such that, at this point, it is important to map this field of research by scientifically identifying and classifying the relevant literature discussing the use of visualization in software evolution. Based on this premise, to better understand the area and to identify the shortcomings of it, we decided to do a systematic mapping study of SEV.

The Mapping Study (MS) reported here follows guidelines from Kitchenham et al. [23] and Petersen et al. [31]. Its main focus is to analyze the SEV area from both the software visualization and software engineering points of view. For that, it provides: (i) a comprehensive survey of software evolution visualization approaches; and, (ii) a classification of the studies using attributes derived from both areas. It collects evidence on how SEV research is structured according to visual paradigms, visual attributes, strategies of analysis, software engineering goals, empirical validation, data sources and metrics employed by the reviewed studies.

The remainder of this paper is organized as follows: Section 2 describes the systematic mapping process undertaken in this study. Section 3 presents the main findings of the study and analyzes them. Section 4 considers the threats to the validity of this research. Finally, Section 5 presents a summary of the work and the directions for future work.

[1] The primary studies that were identified in this mapping study are cited in this work starting with S. A complete list of these studies can be found in the Appendix B.

2. Systematic mapping process

The Mapping Study was used to provide a structure for the type of research and results that have been published in SEV by categorizing them [31].

2.1. Research methodology

2.1.1. Research questions

The purpose of this work is to identify and characterize the body of knowledge of relevant literature discussing the use of visualization in software evolution through the main question: "**What maintenance tasks are current SEV technologies evidently supporting and how do the approaches differ from each other?**"

This question has two main dimensions. The first is concerned with the goals of the SEV approaches. The focus here is to understand the purpose of the proposed SEV. It is the most important dimension of our study. The second dimension intends to identify other differences among the proposed approaches. This paper looks both at information visualization and software engineering attributes to classify those differences. From the information visualization perspective, it looks at concepts such as visual paradigms, visual attributes and mechanisms of interaction used by the SEV approach. From the software engineering perspective, it looks at types of data sources and SE metrics used by the SEV approach as well as the type of experimental evaluation applied to validate it.

The paper also looks at two other important attributes that sit in the intersection of the information visualization and software engineering domain: perspectives and strategies of analysis of the SEV approach. The concepts of perspective [7,30,45–47] and strategy of analysis [S125,S118,29,48,49] were introduced in our previous works. Both are explained latter in this paper.

2.1.1.1. Refining the main research question into specific questions. As explained above, the main question has different facets. So, to address these facets, we derived a set of nine sub-questions that guided the study. These questions define the classification facets of the study, and address topics related to information visualization, software engineering, and their intersection.

Q1. What analysis tasks does the SEV claim to support?

This question aims to identify the goal of the SEV tool, or proposed approach. The visualization of software evolution is generally used to address some software engineering challenge or problem. Examples of those are the identification of: (1) hot-spots of design erosion and code decay [43]; (2) elements that are inducing code decay [11]; and, (3) code smells in the software [26].

Q2. What types of visual paradigms are used by the study?

This question intends to find out the visual paradigms commonly used to visualize software evolution. Visual paradigms are used to create the *de facto* visual scenes (views) rendered on the computer screen. This paper follows the classification of visual paradigms proposed by Keim and Kriegel [21]. They list five main approaches for visualizing large amounts of multidimensional data: Pixel-Oriented, Geometric Projection, Icon-Based, Hierarchical and Graph-based.

Here it is important to remark that more than one view (graph, diagram, or window) can be used to compose a single visual paradigm and, similarly, more than one visual paradigm can be used by a SEV tool to portray software evolution.

Q3. Which types of visual attributes are used by the SEV?

This question seeks to find out the visual attributes (e.g. form, color, movement, and spatial position) that are most used to visualize software evolution. This paper follows the classification of visual attributes proposed by Ware [35]. Visual attributes have different effectiveness in representing real (categorical or continuous) attributes of the software. This effectiveness is also related to visual paradigm and domain.

Q4. Which types of mechanisms of interaction are used?

A common feature of software visualization tools is the use of mechanisms of interaction. According to Heer and Shneiderman [15], information visualization tools should provide these mechanisms to improve the data analysis and user satisfaction. Typical mechanisms of interaction are zooming, panning, brushing, detailing on demand, dynamic filtering, etc. This is especially important when one has to cope with large data sources.

Q5. What data source is used?

Due to the complexity of software evolution, there are different types of data that can be used to visualize and analyze evolution. As some examples, historical information can be gathered from: (i) Source Code Management (SCM) systems, like CVS, Subversion, and GIT; (ii) Bug (or Issue) Tracking System (BTS), like Jira, Bugzilla, and Mantis; (iii) project documentation, like different version of requirements or design documents; (iv) generated or manually produced data, like manually produced or heuristically generated feature mapping files [29]; and (v) the different versions of the source code itself.

Q6. Which types of metrics are visualized?

Measurement is necessary to characterize software projects, products and processes [37]. Metrics are commonly used in software visualization to decorate and enhance the expressive power of visual scenes. Metrics are extracted from the available data sources and associated with visual attributes of the created visual scenes. Typical SCM metrics are number of commits, number of authors, logical coupling [13], type of change made (as in adding, deleting, or modifying a software element) [22]. Typical BTS metrics are number of issues, number and types of bugs, number of bugs open and closed, etc. Typical source code metrics are size, complexity, coupling, cohesion, and object-oriented (OO) metrics [8,26].

Q7. What type of evaluation is done?

Buckley [5] reported that there is little empirical evaluation in the software visualization area. This paper investigates which types of validation have been reported in software evolution visualization studies, to see if this can also be confirmed in this subarea of SoftVis.

Q8. What types of perspectives are used by the study?

A perspective is concerned with the way one sees the software to execute a certain software engineering task [7,45–47]. The perspective usually considers the point of view of the professional interested in performing a certain task, for example, the perspective of the analyst, the perspective of the programmer, etc. A perspective is composed by a set of coordinated views designed to represent a group of properties of the software. These properties can be, for example, the modular structure (package, class, and method), inheritance (class A extends class B), and dependency (class A uses class B) of software systems.

Q9. Which strategy of analysis is used?

Our previous work on SEV has been investigating different ways to analyze software evolution [S125,S118,29,48,49]. During this work, we identified that software evolution can be visually analyzed using different strategies. A strategy of analysis of software evolution defines how the changes of a software artifact are presented for analysis. The representation of the evolution can be portrayed in different ways that may facilitate, or hinder, the understanding of phenomena associated with the changes.

During our mapping, we classified the studies in the following five strategies: Differential Relative, Differential Absolute, Temporal Overview, Temporal Snapshot, and Temporal Accumulative Snapshot. These strategies are explained below.

2.1.2. A brief explanation of the goals of the SEV approaches (RQ Q1)

Software visualization efforts can be categorized in two main dimensions: (i) one focuses on computer graphics issues, dealing with scalable visualizations or displaying large amounts of data in an elegant, comfortable and efficient (easy to interpret) way; and, (ii) the other focuses on pragmatically helping software engineers to achieve software engineering tasks. We believe software visualization approaches must always be well aligned with the second (software engineering) dimension, even when a lot of effort is put in the first (graphical) dimension. This is the same as saying that software visualization should always be goal-oriented [3], pragmatically trying to achieve software engineering tasks. So, for us, the approach goals should drive the definition of software visualization tools and not the other way around.

We used Basili and Weiss [4] goal template to try to thoroughly capture the software analysis goals expressed in the SEV papers. This template has typically four facets: point of view, object of analysis, focus and purpose. The object of analysis is the entity of interest in the analysis. The purpose is usually classified in characterizing, understanding, evaluating, controlling, improving or predicting something. The focus is the main attribute of interest in the object of analysis. And, the point of view describes the user group of the analyzed information. Those facets had to be adapted to the SEV domain.

The *point of view* describes the user group of the analyzed information. Software maintainers or software managers can use many SEV approaches, but most papers did not explicitly mention what is the target audience for the approaches they propose. For sake of simplification of our mapping, we decided not to gather the point of view of the SEV approaches, and assumed them to be either for *software maintainers* or *software managers*.

The *object of analysis* is the entity of interest in the analysis. We needed a classification schema for the SEV objects of analysis, but we did not find one in the literature. We decided to adopt a schema from a related field and expand it to SEV. The most valuable source for this purpose was the taxonomy proposed by Kagdi et al. [22]. They classified studies of mining software repositories (MSR), which are, in several dimensions, very related to SEV.

Focus and *purpose* are usually tangled together in SEV analysis. They explain why SEV is being used (e.g. to understand (purpose) software changes (focus)). Like before, we created a classification schema based on the Kagdi et al. [22] and the applications of SEV we found in the literature.

In summary, in order to capture the goal of the studies, we record "what" the study proposes to visualize and "why" the study is visualizing it. "What" is related to the object of analysis. "Why" is related to the focus and purpose. Together they define the data that should be gathered to the SEV. Both classification schemas will be presented in Section 3.3.1, when we present the mapping study results.

2.1.3. A brief explanation of the strategies of analysis in SEV (RQ Q9)

As mentioned previously, we have identified differential and temporal strategies for visualizing software evolution. The differential strategies analyze the evolution considering two versions per time, while the temporal strategies consider all available versions at once. These strategies are explained below:

Differential. This strategy takes into consideration only two versions at a given time to analyze the evolution. The differences are calculated between the entities of two selected versions. Normally, the two versions can be dynamically selected from any available version in the software history. The differential approach can portray complex visual structures painted with the differences between the chosen versions. A key feature of this type of strategy is the possibility of interactively selecting the two versions to be compared. The Differential strategies can be specialized in Relative and Absolute.

Differential Relative (DR). This strategy is used to represent the growth or reduction related to a property of a software module. If a software module *a1* grew by a value *x* for a given metric *m*, and another module *a2* grew by a value *y* for the same metric, such that *y* is greater than *x*, then the DR representation should visually highlight that *a2* grew more than *a1*. The definition of the reduction occurs analogously. As state by the name, in this type of analysis, the concept of variation (increased or decreased) is relative among the software modules. For example, a module that increased by only five lines of code from one version to another may be highlighted as one with a significant growth with respect to this metric, if all other modules have change much less than that. Note that, in this case, five lines of code alone may not represent a large growth, but it stands out relative to the other modules of the software being analyzed.

Differential Absolute (DA). The absolute strategy is useful when the definition of growth or reduction does not apply. It is concerned with identifying and representing discrete events, such as entities properties that, for example, have appeared or disappeared from one version to another of the software. As an illustration, consider that from a version i to a version j, $i < j$, a software module *a1* starts (or ceases) to realize a particular feature f of the project. A Differential Absolute strategy would then visually flag this event in the differential view.

Temporal. This strategy portrays the evolution considering all the versions available for analysis. Given n versions $v1, v2, \ldots, vn$, (or a sizeable sequential subset of them), this analysis takes into account everything that has happened from version $v1$ to version vn taking in consideration all the intermediate versions. It helps, for example, to analyze patterns of changes between different metrics over the software evolution cycle. The temporal strategy can be specialized in Overview, Snapshot and Accumulative Snapshot.

Temporal Overview (TO). This strategy of analysis, unlike DR and DA, produces an overview of the evolution of software throughout its evolution lifecycle. The TO strategy shows all the values of evolution of a given metric for all software entities being analyzed. This complicates the presentation as we have to show in a single visual scene many entities (e.g. software modules) together with the temporal evolution of one or several of its attributes (e.g. size).

Temporal Snapshot (TS). This strategy shows the current state of the software (all entities being analyzed) at some time in its history. It is a snapshot of the software in a given moment in time. This strategy requires a good timeline interaction mechanism to allow the user to easily navigate through the project history. Frequently, it uses animation to illustrate changes over the software history.

Temporal Accumulative Snapshot (TA). This strategy is a subtype of the TS strategy. It takes in consideration the absolute value of changes to analyze the evolution of the software. It is frequently used to analyze software churn [14]. To illustrate its workings, consider a software with three versions $<vk, vl, vm>$, such that $k < l < m$. Suppose a metric M with the following values for a given software module *A1*: $Mk = 5$, $Ml = 3$, and $Mm = 9$. At the snapshot vm, the module under analysis has the cumulative variation of $|Ml - Mk| + |Mm - Ml| = |3 - 5| + |9 - 3| = 8$. This measure is a snapshot of module *A1* at the moment vm, but it captures its history of churning according to the chosen metric M.

In [18], the authors presented a study to understand design patterns decay, grime and rot in evolving software systems. They observed software properties (e.g. design anti-patterns, afferent or efferent coupling) that increase or appear along the software history. The *Differential* strategies could be used to visually represent these events considering two versions per time. On the same token, the *Temporal* strategies could also be used, in this case, to visualize the whole evolution of those properties at a glance.

2.1.4. Inclusion and exclusion criteria

We adopted the following inclusion and exclusion criteria to select studies suitable to our work.

Inclusion Criteria. In order to be included in our work, a study needed to explore a theory, a practice, an approach, a strategy or issues related to the use of software visualization to handle software evolution. There is an intrinsic relationship between software visualization and tool implementation, however if a study just proposed a new visual way to portray software evolution data, without any implementation, it was also considered. In addition, the most complete version of the work was used, if the study was published in more than one venue. However, if the paper was an extended version of some work, we considered both versions. We decided to do that because it is common in software visualization area to see a proposal of a new visualization approach in the main conference track and its actual implementation in tool demonstrations track. These papers are usually different regarding to at least one of our research questions: the type of evaluation.

Exclusion Criteria. The work excluded those studies that do not address software evolution visualization. In particular, studies that analyze software evolution data, and uses charts to discuss their findings were excluded, because in spite of being about software evolution analysis, they are not about software evolution visualization (e.g. [10]). According to [9], "software visualization is the art and science of generating visual representations of various aspects of software and its development process". In contrast with this definition, these works did not develop visual representations to portray the evolution. They extract the evolutionary data and use charts, from other tools (e.g. spreadsheets) to represent the data. Frameworks, surveys and secondary experimental studies were excluded, since they report the approaches from others. We found papers presenting research group projects (e.g. [1]). They were also removed, since they only present the software evolution visualization as a research area of the group. We excluded papers that aim to help users to do maintenance activities, and consequently to evolve the software, but, for that, provide only views (considering metrics and relationships) of one version of the software [36]. On the same token, we also excluded papers that stated that their approach could be used to do software evolution analysis, but whose main focus was single version analysis (e.g. [24]). These papers usually describe how the approach can be used to visualize the evolution, but only as a proposal. Papers that say the approach can be used for software reengineering, but just describes a software architecture were also removed (e.g. [34,19]). We also removed works published only as gray literature such as thesis and books. Finally, we also excluded studies that are only available as abstracts or slide presentations.

2.1.5. Classification scheme

We followed four directions to classify the papers: (i) recovering meta-information of the studies; (ii) analyzing the studies from a software visualization point of view, by using some classifications that highlight important features of the approaches and tools [32,21]; (iii) from a software engineering point of view, by classifying the papers considering the goal of the study, the data sources, the metrics, and the type of evaluation performed (or not); and (iv) from the way software visualization is used to execute the software engineering tasks, by classifying the papers according to the perspectives and strategies of analysis used.

In order to answer the questions listed in Section 2.1.1 we adopted the following classification facets:

– *Metadata of the studies.*

We recovered metadata from the papers. This allowed us to identify **venues** where the studies were published, **authors** and **affiliations** of the papers, **type of the papers** (short or full papers), **type of venues** (workshop, conference or journal), and **years** of publication.

– *Software Visualization (Sub-questions 2, 3 and 4).*

We classified the **visual paradigm** according to [21]. They defined five types of paradigms: Pixel-Oriented, Geometric Projection, Icon-Based, Hierarchical and Graph-based. The **visual attributes** (such as color, form, and geometric position) used to present software attributes were also identified. We also recorded if the approach is based on multiple views (**multi-view**) or not. Lastly, we also tried to identify the **mechanisms of interaction** used by them.

– *Software Engineering (Sub-questions 1, 5, 6 and 7).*

Based on the *Goal Question Metric* (GQM) approach [4], we recorded the main **goal** of the visualization approach. And, associated with this goal, we recorded which types of **data sources** were used by the SEV tool, which **metrics** were used to characterize the analyzed software artifacts, and the type of **evaluation** performed to validate the approach.

– *The way software visualization is used to execute software engineering tasks (Sub-questions 8 and 9).*

We recorded which **perspectives** and **strategies** were used to analyze software evolution. These two classifications facets represent the intersection between software visualization and software engineering. They are concerned with the way software visualization is used to perform software engineering tasks. Perspective is a concept related to software visualization in general, however, strategies are specific to software evolution analysis.

Fig. 1 provides an overview of the facets pointing out the intersection between SE and SV areas.

2.1.6. Data extraction

We used a data extraction form designed to gather the desired information with the facets described above. Two researchers read each paper. If the researchers disagreed in a classification, or information gathering, a third opinion was used to resolve differences and reach consensus.

2.2. Conducting the review

2.2.1. Conducting the search of primary studies

In consonance with [23], the first step in conducting the mapping study is to find candidate primary studies for analysis. For that, we searched the available software engineering literature.

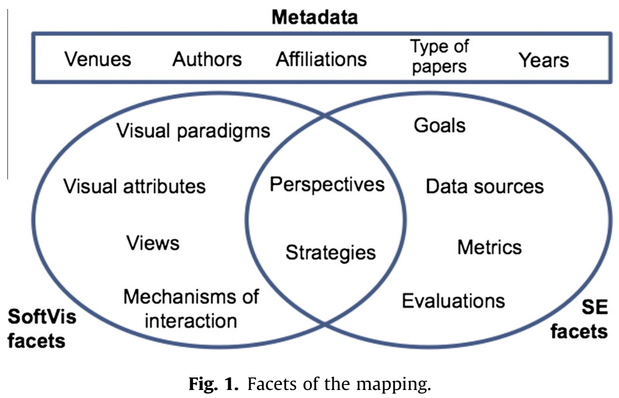

Fig. 1. Facets of the mapping.

To define the search terms, we selected the keywords: software, evolution, and visualization. Later, we identified alternative spellings and synonyms for these words, and used OR and AND operators to assemble search strings. The complete list of search strings used in this work is shown in Table 1.

The search process used did not involve manual searches. The reasons for that are: (i) there are no conferences or journals specifically dedicated to software evolution visualization; (ii) the two main workshops in the area of software visualization are indexed by digital libraries (see Appendix A); (iii) software evolution visualization can be used for different software engineering goals, so to be fair and replicable, a manual search would have to cover all existing work presented in software engineering conferences or journals; and (iv) we selected six digital databases which, to our knowledge, index the most important conferences in the software engineering area.

The search covered all papers published up to 2011 (including). The search strings in Table 1 were calibrated and applied in each digital database (see Appendix A). The title and abstract of the searched papers were then analyzed to check if they indeed dealt with software evolution (Filter 1). The resulting set was then analyzed in more depth to check if they contained the information we proposed to collect (Filter 2). To complement the search process, we applied a ‘snow-balling’ process [6], in which the references of the identified papers were analyzed to expand the base of identified papers. Fig. 2 illustrates this search process. It indicates that the first search returned 8945 papers, the title and abstracted analysis (Filter 1) reduced the list to 516 papers, the in-depth reading (Filter 2) reduced the list to 125 papers, and the backward snow-balling expanded it to 146 papers, our final (primary studies) universe of analysis. During the snow-balling process, we looked at the title of each reference in the 125 papers (list after Filter 2). If the paper title made any reference to software evolution visualization, we would repeat the last three steps presented in Fig. 2, for the selected paper.

2.2.2. Looking at our paper sources

Table 2 shows the number of studies in each step and per each digital library. It can be observed that Scopus and Engineering Village returned a large set of papers. In these libraries, we found many papers from other areas that had nothing related to our study, like “**software** that **visualizes** the **evolution** of the species”. Note that the table shows the amount of papers with and without intersection. Filter 2 was the step in which we definitely removed the intersections. After both filters were applied, we had 125 papers, and after the snow-balling we ended up with 146 primary studies.

Table 1
Search strings.

(Software OR System)
AND
(Evolution OR Evolving OR Evolve)
AND
(Visualization OR Visual OR Visualisation)

Table 2
Number of paper per step and per digital database.

Digital databases	Search	Filter 1	Filter 2
IEEE	865	120	83
ACM	617	90	58
DBLP	56	28	26
Scopus	5274	156	70
Engineering Village	1564	89	64
Science Direct	569	35,693	7
Total	8945	516	308
Total without intersection		270	125

From Table 2, one can see that the IEEE digital library returned most of the relevant papers (83 of 146 primary studies, 56.85%). Table 3 identifies the number of papers that were found exclusively on a given digital library. From Table 3 we can see that all papers found in the ACM DL were also found in at least another digital library.

3. Results

This section presents the main findings of the data extraction activities. It discusses each research question listed in Section 2.1.1, but to improve understanding, we grouped the findings according to the classification scheme listed in Section 2.1.5.

3.1. Metadata of the studies

This section focuses on classification of the origins of Software Evolution Visualization studies.

3.1.1. Publication venues

The software evolution visualization papers are published in many different venues. We identified 56 venues where the papers were published. Most of the venues are from Software Engineering domain, usually software comprehension, maintenance and evolution events. Fig. 3 portrays the top 10 venues for the area, where ICSM stands for the International Conference on Software Maintenance, WCRE for Working Conference on Reverse Engineering, VISSOFT for International Workshop on Visualizing Software for Understanding and Analysis, IWPSE for International Workshop on Principles of Software Evolution, CSMR for European Conference on Software Maintenance and Reengineering, SOFTVIS for International Symposium on Software Visualization, ICSE for International Conference on Software Engineering, MSR for International Workshop on Mining Software Repositories, TSE for IEEE Transactions on Software Engineering, and JSME for Journal of Software Maintenance and Evolution.

Table 3
Number of papers exclusively per digital database.

IEEE	ACM	DBLP	Scopus	EV	SD
22	0	2	5	4	1

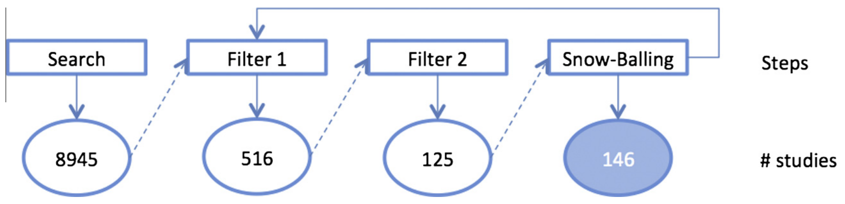

Fig. 2. Number of studies in each step of the Screening.

Fig. 4 shows the venues classified by the type and the number of papers.

3.1.2. *Years and type of the publication*

Some authors consider software evolution visualization as an emergent area [44]. As the area matures, the papers should move from short papers to full papers. From the 146 primary studies, 111 are full and 35 are short papers. Fig. 5 shows the distribution of the publications per type and per year.

The top of Fig. 6 presents the number of papers per type of venue and per year for all venues. As can be observed, the number of papers grew strongly in 2004, 2005 and 2006, peaking in that year, and is steadily decreasing since then. The number of journal papers kept almost the same during the investigated years. Overall, the number of workshop and conference papers increased during the last decade. The bottom of Fig. 6 shows type of paper (short or full) for the top 10 venues presented in Fig. 3. As can be observed, for those vehicles, the number of short papers has increased since 2006. Since then, the number of short and full papers is well balanced for those venues.

3.1.3. *Authors*

The purpose of this topic was to identify the most influential researchers in the area. We found that 184 different authors published SEV papers, showing a wide interest on this subject in the software engineering research community. Fig. 7 shows the top 10 authors and number of papers they published in the area.

We also decided to investigate how often these studies referenced each other. Table 4 shows the top 10 referenced papers. The Evolution Matrix [S54], by Lanza, is the most influential paper, referenced 35 times by the other 145 analyzed primary studies.

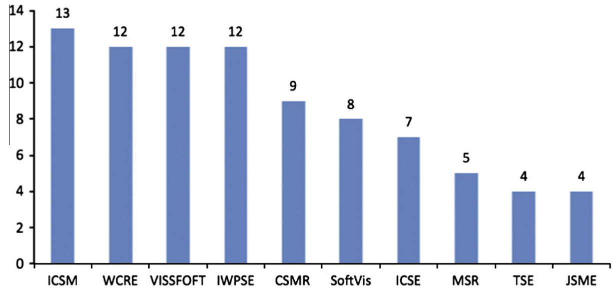

Fig. 3. Top 10 venues where the SEV papers were published.

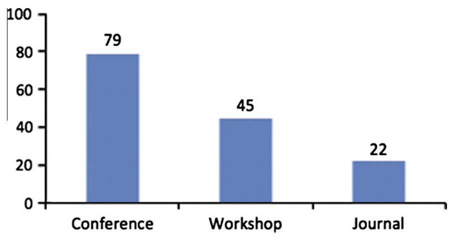

Fig. 4. Number of papers per type of venue.

3.1.4. *Affiliations*

Fig. 8 shows the country of affiliation of the authors of the papers. 52 studies (35.62%) are from Switzerland, followed by USA and Netherlands, with 20 papers each, and Canada with 19 papers.

3.2. *Software visualization*

To answer research questions (RQ) Q2, Q3 and Q4, we now focus our analysis on the information visualization aspects of the software visualization tools.

3.2.1. *Visual paradigms (RQ Q2)*

As can be observed in Fig. 9, Graph-based, Hierarchical and Geometric Projections are the most used visual techniques, with 66, 54, and 53 studies, respectively. Following this list, Pixel-oriented technique was presented in 37 studies and Iconographic techniques in only 23 studies.

From the totals, one can observe that most studies use more than one technique to visualize the data under analysis.

3.2.2. *Visual attributes (RQ Q3)*

Fig. 10 shows the number of studies per the most common visual attributes. According to the collected data, the color is the most frequent visual attribute used in the studies. 137 studies (93.84%) use this attribute. This can be explained, since color and spatial position can be immediately observed and processed by the human visual system [28]. They are also fairly easy to implement in software tools.

3.2.3. *Views (RQ Q3)*

Fig. 11 shows that 77 papers (52.74%) describe visualization approaches that use only one view to display graphical information. This was surprising, since we expected that SoftVis authors would prefer to use more than one view in order to produce different perspectives of the analyzed software. Unfortunately, we could not observe in this mapping study if these one view approaches reach better, similar or worse results than the multiple view approaches. The area needs further studies to investigate this.

It should also be noted that in spite of the fact that most of the studies use only one view, most still manage to use more than one visual paradigm, see Fig. 9. This means that many authors propose views that combine paradigms. For example, they may combine

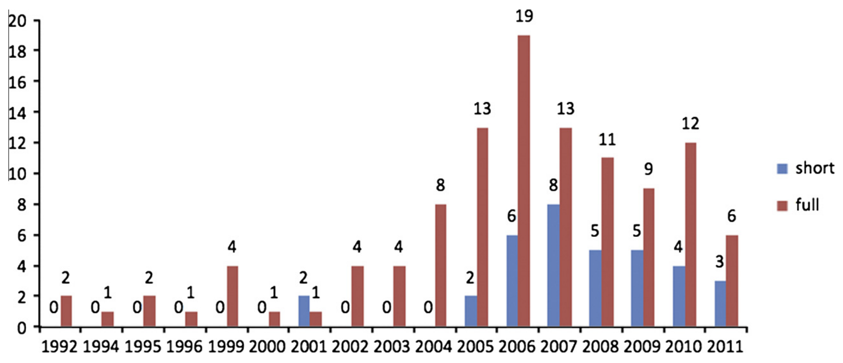

Fig. 5. Number of studies per type of publication per year.

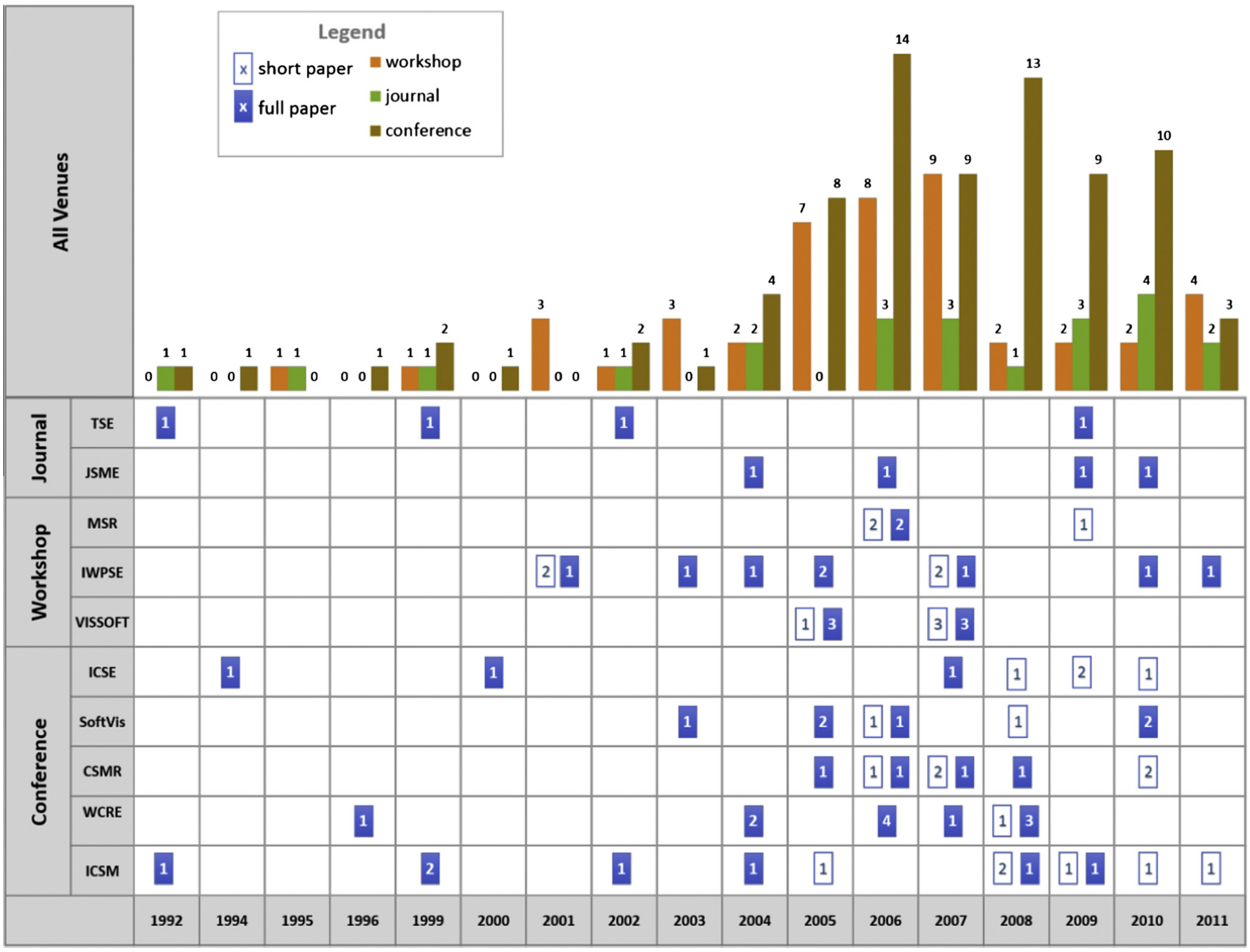

Fig. 6. Number of paper per type of venue, type of paper and year.

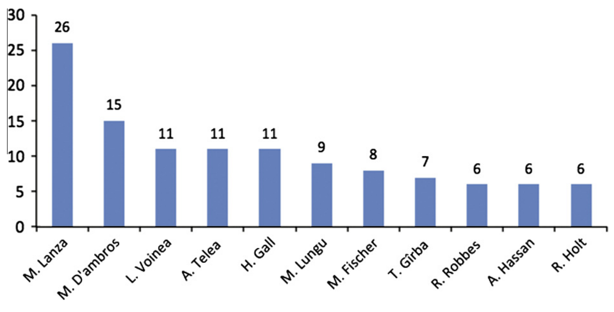

Fig. 7. Top 10 authors.

icons and graphs in the same view in order to build a richer visual metaphor.

3.2.4. Mechanisms of interaction (RQ Q4)

Interaction mechanisms are controls for compositing, dynamically modifying, and navigating between views of the software. We decided to investigate this facet in our mapping due to the importance it has in the information visualization, in general, and software visualization, in particular. This holds notably true when the analysis deals with large amounts of data like we do in SEV. Interaction mechanisms are used to dynamically filter, select, pan, and brush through the data being visualized. The response time between interactions and re-rendering of visual scenes are instantaneous (or animated) for all practical purposes. Strategies like *temporal snapshots*, *differential relative* and *differential absolute* only make sense when combined with good interaction mechanisms.

Unfortunately, evaluating interaction mechanisms from written papers was not an easy task. Few papers, like [S75], describe all the tool's interaction mechanisms. Many studies do not discuss at all the interaction mechanisms present in their SEV tools, possibly because interaction mechanisms are many times seen as an implementation issue.

We investigated some tools by visiting the website provided in the paper. We found that several interaction mechanisms are indeed usually available in the SEV tools. Unfortunately, as we said before, the written reports were incomplete to say the least. To get around this kind of detailed information in our mapping study, we only counted the interaction mechanisms that were explicitly cited in the papers, or that could be observed in the pictures contained in them. Fig. 12 shows the number of studies per mechanisms of interaction we found. The most common mechanism found was Selection (70–47.95%).

It is important to remark that these are not great numbers. To deal with the large amount of data needed in the context of software evolution, some mechanisms are a must. Navigation helps to go through the data and the different versions of the software. Filter, querying, zoom and sort can be used to organize or restrict the amount of data displayed in the view canvas. We observed that

Table 4
Top 10 referenced papers.

Paper	Number of references
The Evolution Matrix: Recovering Software Evolution Using Software Visualization Techniques [S54]	[35]
A System for Graph-Based Visualization of the Evolution of Software [S51]	[29]
Visualizing Software Release Histories: The Use of Color and Third Dimension [S57]	[27]
SeeSoft – A Tool for Visualizing Line Oriented Software Statistic [S127]	[24]
CVSscan: Visualization of Code Evolution [S103]	[21]
Studying Software Evolution Information by Visualizing the Change History [S48]	[20]
Visualizing Multiple Evolution Metrics [S45]	[20]
Exploring Software Evolution Using Spectrographs [S46]	[17]
Characterizing the Evolution of Class Hierarchies [S88]	[14]
Software Bugs and Evolution: A Visual Approach to Uncover Their Relationship [S38]	[12]

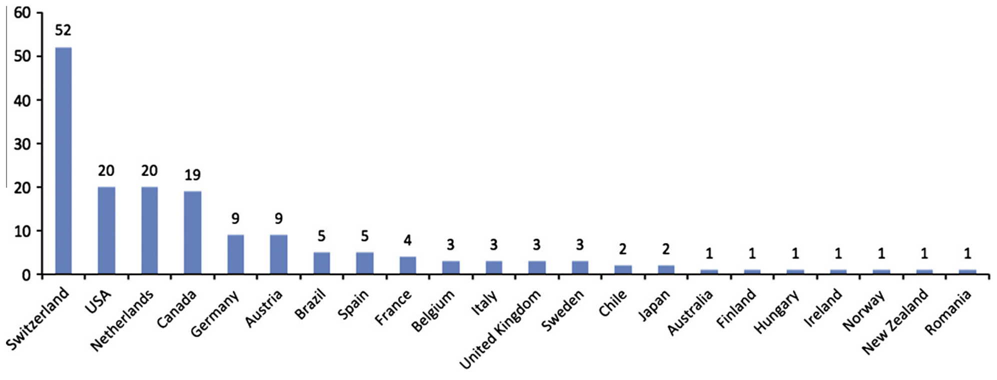

Fig. 8. Studies per country.

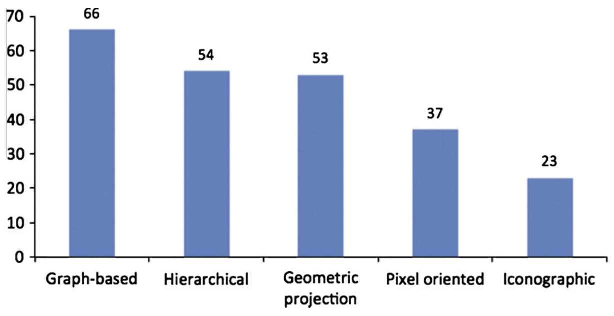

Fig. 9. Number of studies per visual paradigms.

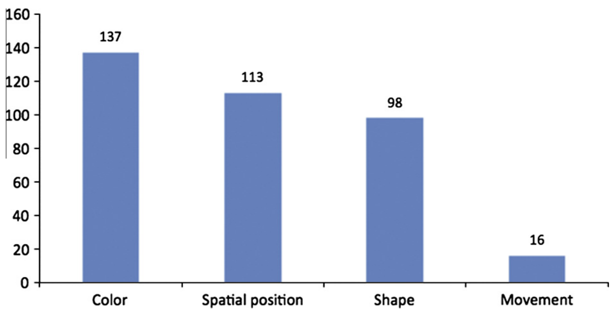

Fig. 10. Number of studies per the most common visual attributes.

the visualization mantra “Overview first, zoom and filter, then details on demand” [32] is not always applied by the implementers of current SEV tools.

3.3. Software engineering

To answer research questions (RQ) Q1, Q5, Q6 and Q7, this section focuses on the classification of the studies from the software engineering point of view. We organize the studies according to their *goals*, *data sources*, displayed information (*metrics*) and *evaluation* approaches.

3.3.1. Goal (RQ Q1)

This paper uses a classification based on Basili and Weiss [4] goal template to capture the software analysis goals expressed in the SEV papers. This template originally had four facets: point of view, object of analysis, focus and purpose. The mapping results for the SEV goals are presented below. As explained in Section 2.1.2, they are based on three of those facets: object of analysis, purpose and focus.

3.3.1.1. Objects of analysis. As explained in Section 2.1.2, we based our object of analysis classification on the classification by Kagdi et al. [22]. For that, we grouped the SEV approaches on the categories presented in Table 5.

Fig. 13 presents the number of studies per each category. Some studies were classified in more than one category.

3.3.1.2. Focus and purpose. Kagdi et al. [22] also proposed 10 evolutionary mining task categories, and classified visualization in a “change comprehension” category. In terms of GQM’s goals they are saying that SEV is mostly used to *understand* (Goal Purpose) *changes* (Goal Focus).

In fact, most of the studies we analyzed present approaches that tell the history of a software artifact, showing what happened to it since its creation. For example, many works try to help users to “gain insights in the code (e.g. [S26, S44])”; “to cope with a large amount of data (e.g. [S42])”; “helps software engineers build knowledge about their systems [S63]”; “observe the change along the software evolution. [S99]”; “help in understanding how a software product has evolved since its conception [S100]”; and “this representation highlights the critical times—when major changes seem to have happened—in the evolution of the whole module [S56]”.

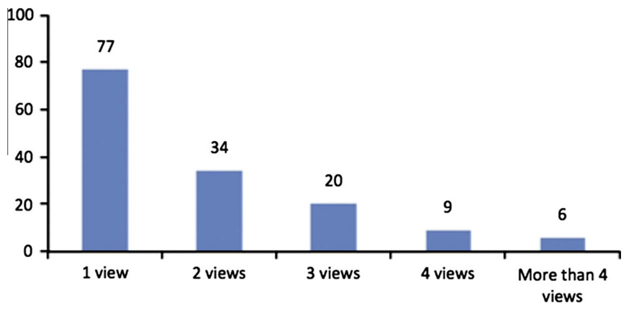

Fig. 11. Number of views per number of studies.

However, there are some studies that do not have a specific software engineering purpose. They mainly aim at discussing the visual resources, presenting them as a powerful means to represent software evolution visualization data [S33]. We classified the purpose of these studies as "provide powerful and scalable visualization".

Table 6 shows the categories we used during the classification of the SEV purposes. Again, some studies were classified in more than one category. But it must be noted that we only put a study in a category, if the purpose of the study was explicitly stated in the paper.

Fig. 14 shows the number of studies per each category presented in Table 6.

3.3.2. Data sources (RQ Q5)

Data sources are essential to the data analysis tasks and in the software evolution domain. They are varied, and usually large in size. Fig. 15 summarizes our findings. The main data source for the goals listed in the previous section is the Software Configuration Management systems (SCM). Examples of such systems are CVS, SVN and GIT. These systems maintain a history of changes in all files, and control the authors responsible for the modifications. SCM data include information such as commit messages, commit time, commit author, and commit type (added, modified, or deleted). With the growth of SCM and open source systems, studies using this type of data have become more frequent. We found in this mapping that 96 studies (65.75%) use SCM as one of its data source.

One of the shortcomings of these data sources is the level of granularity of the recorded changes. They normally track changes at the file level. However, one is usually interested in tracking changes at more specific parts of the code (e.g., changes made to

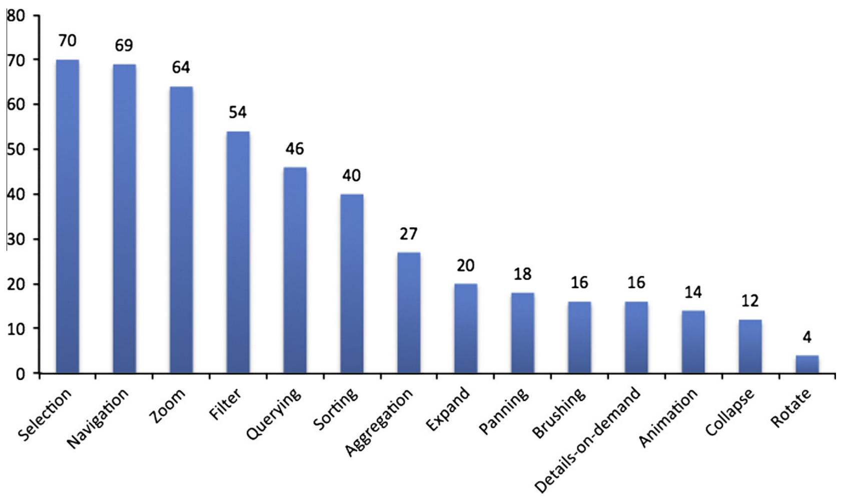

Fig. 12. Number of studies per mechanism of interaction.

Table 5
What the software evolution visualization approaches visualize (Object of Analysis).

What the SEV visualizes	Description
Behavior	Visualize dynamic information, as execution traces, profiling executions, etc.
Code Clones	Visualize the clones in the source code and handle their evolution
Data structures	Focused on data structures of the programs
Defects	Any type of defects, generally extracted from BTS
Developers Activity	Aimed to visualize which developers have worked on the project, or where developers have worked
Developers Dependency	Aimed to visualize how developers work together, how they communicate among them, or which files are owned by more than one developer, etc.
Exception Handling	Focused on handling exception flow on programs
Ecosystem Change	Visualization of ecosystems
Features	Focused on features, concerns or functionalities of the source code
Mail List	Visualize the information contained in developers mail list
Metrics	Focused exclusively on the evolution of metrics. Normally expressed in Kiviat diagrams or Parallel coordinates
Requirements	Focused on the requirements of the program
Software Architecture	Focused on models, structure, or UML diagrams
Source Code Change	Visualize the changes related to the source code itself. Metrics of size are common used by these studies
Source Code Comment Change	Enrich commit comments using software visualization
Source Code Dependency	Focused on analyze how the program is coupled
System Dependency	Dependency between systems
Topics	Visualize collections of words that co-occur frequently in a corpus of text

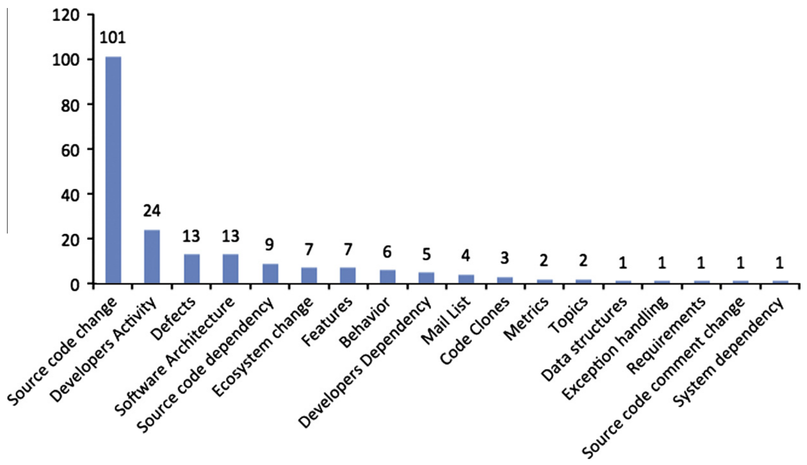

Fig. 13. Number of studies per "what" categories.

Table 6
Why the software evolution visualization approach was applied (Purpose).

Purpose of SEV	Description
Change Comprehension [22]	To understand how software has been changed in a given period of time (e.g. [S1]), identify evolutionary patterns (e.g. [S38]), or identify stable parts of the software (e.g. [S32])
Change Control Enhancement	To enhance the control of the modifications to have a better comprehension of the changes
Change Prediction	Predict which module will change (e.g. logical coupling), candidates for refactoring (e.g. a class with high coupling [S15, S29, S45]), help to identify unstable elements that need to be rewritten [S51], or re-engineering [S93], impact analysis [S97], estimating effort [S128]
Contribution Analysis [22]	Developers activity, who worked on the software
Defect Classification and Analysis	Comprehension, prediction, of defects. Identify which module will be buggy
Development Communication [22]	How, or which developers communicate among them
Quality Improvement	Provide quality improvement
Process Improvement	Help the users to take decisions, that affect the process improvement
Powerful and Scalable Visualization	Where the goal is to improve the software visualization area, some studies highlight that the proposed visualization are scalable
Re-documentation	Re-document the system
Reverse Engineering	Help to do a reverse engineering
Software Reuse	Help users to reuse the software
Identification of Anomalies	Code smells (e.g. god class [S19]), design erosion or software aging [S23], signs of structural decay or the spread of cross cutting concerns in the code [S29], point out and assess structural stability [S31], architectural deterioration [S46], violations to the architecture [S139]
Dependency among Projects	Identify which projects depend on other projects

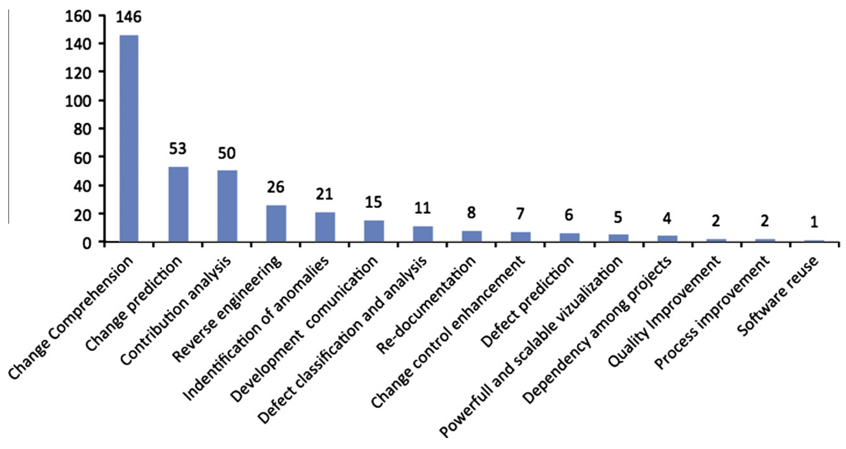

Fig. 14. Number of studies per "why" categories.

a method). To overcome these shortcomings, the SEV authors have taken two approaches: (1) keep track the changes in a lower level of granularity ([S69, S123, S129, S137, S146]); and (2) parse the source code itself to extract more detailed information directly from it. The second option is used on 75 (51.37%) of the studies we mapped.

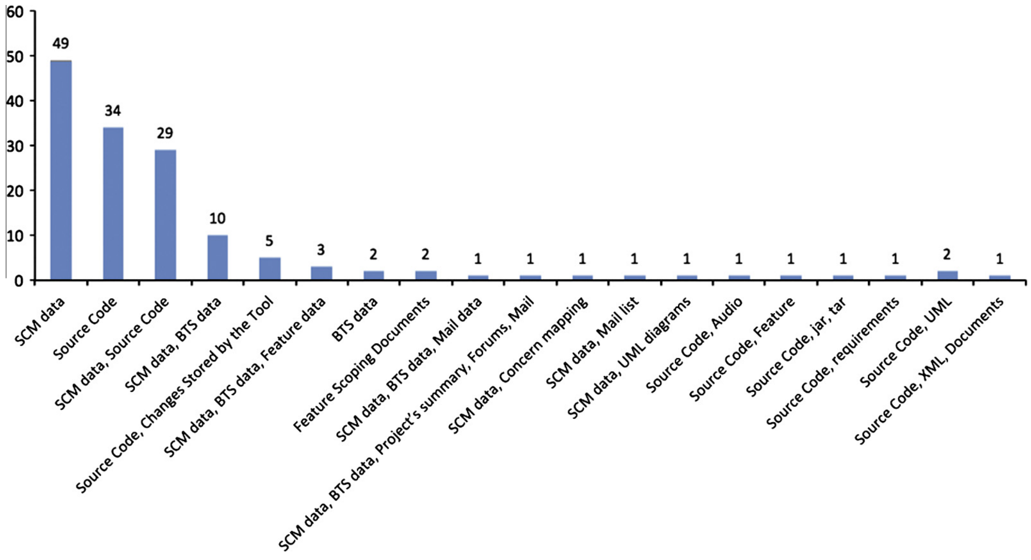

Fig. 15. Number of studies per type of data sources.

Many tools tends to use the source code as its main information source, probably because the source code is the most formal and unambiguous artifact developed and handled by humans during the software development process [S73,S125]. Source code is also well suited for the extraction of a large number of popular software engineering metrics. On the down side, source code analysis usually makes the SEV tool language specific.

A smaller number of tools also uses Bug Tracking Systems (BTS) as a data source. BTS records maintenance and changes occurrences over the software lifecycle. We also found some unfamiliar data sources. The most unusual, from our viewpoint, was the use of audio records [S72].

Fig. 15 also indicates that several approaches (41.78%) make joint use of different data sources. The most recurrent data source combinations were SCM data + Source code (19.86%) and SCM data + BTS data (6.85%). As an example, [S14, S20, S27, S38] are able to interpret the evolution by crossing information from BTS and SCM repositories. Off course, in these cases, the selected data sources should be linked in some way to make the interpretation feasible.

Fig. 15 is complex because it has many data source combinations. To better analyze the data source scenario, Fig. 16 focus on the three main data sources and their intersections. It shows a Venn diagram of the studies that used BTS, SCM, and Source Code (SC). All other data sources are eliminated from this diagram. Combinations are simplified to the core three sources. For example, the study that combined SCM, BTS and Mail data in Fig. 15 is counted as one SCM and BTS study in Fig. 16. The empty segments in the Venn diagram correspond to a zero value.

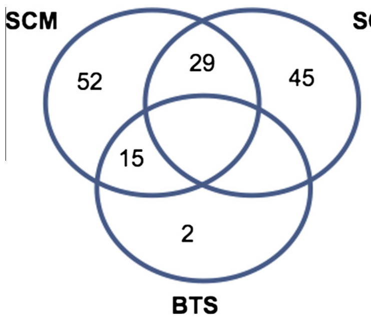

Fig. 16. Looking at the three main data sources.

The main insights that come out of Fig. 16 are that there is no study combining BTS data and Source Code, and that SCM data have been the key data source for software evolution visualization.

3.3.3. Metrics (RQ Q6)

Software entities have attributes, characteristics or properties that we want to measure. Metric sets establish how these attributes, characteristics or properties are measured [12].

We found in this study that 114 studies (78.08%) use more than one metric in their approach. This is expected, since most aspects of software processes and products are too complicated to be captured by a single metric [2]. We also observed that there is a set of metrics that is used the most: coupling, cohesion, size, number of commits, number of authors, and number of bugs. And, for objected oriented metrics, the studies usually use the metric defined by Chidamber and Kemerer [8]. We also noticed that some studies do not explicitly report all the metrics they use.

Fig. 17 shows the most common metrics we have found in the mapping study. Physical Coupling and number of commits are the most used, appearing in 36 studies (24.66%) each one. Considering any type of coupling (physical and logical), we found 56 studies, or 38.36% of the studies.

As it should be, metric usage is strongly related to the goals of the approach. The *object of analysis* and *focus* of the study usually determine the metrics. For example, if the visualization helps to detect clones [S1], a possible metric is number of clones.

3.3.4. Evaluation (RQ Q7)

Empirical validation of technologies has increased significantly in the software engineering domain during the past 15 years. However, experimentation is especially difficult in cognitive and perceptive activities. Software visualization is one of these cases. It aims to improve the software comprehension and is very much dependent on the person realizing the task.

Fig. 18 summarizes our results with respect to empirical validation of SEV approaches. We found basically four categories of evaluation in our mapping study: (i) the authors used the proposed tool over a software object as a feasibility study (106–72.60%); (ii) subjects used the tool in case studies and give feedback to the authors (10–6.85%); (iii) the authors performed a check list comparison among their tool and another related tool (1–0.68%); and, (iv) the authors ran a controlled experiment, in which a formal

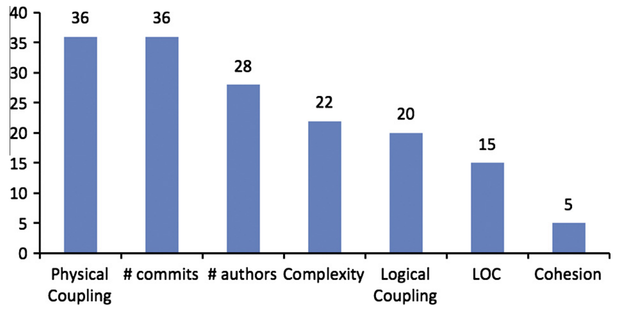

Fig. 17. Commonly used metrics.

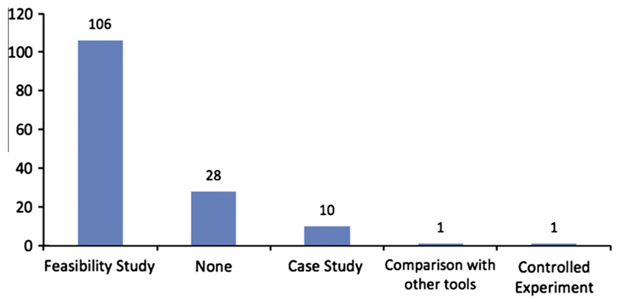

Fig. 18. Number of papers per type of validation.

experimental evaluation is applied (1–0.68%). Finally, 28 studies (19.18%) had no validation.

Fig. 18 summarizes the findings showing that most approaches have performed a feasibility study and many performed no validation at all.

Fig. 19 shows how these studies are distributed according to the artifacts being analyzed. Most performed studies over open software systems (OSS). Fig. 19 also indicates that many of the OSS and Academic studies utilized more than one artifact.

We also found some variations in the feasibility studies done over open software systems. Like in [S17], where the authors used their tool over an OSS project and later confirmed the results with the developers of the analyzed systems, or in [S49] and [S58], where the authors applied the tool and checked its results against the documentation of the project.

3.3.5. Goals × metrics × evaluation

Fig. 20 summarizes the main results of this section. It uses a bubble chart to display *goals versus most commonly used metrics* and *goals versus evaluation*. While looking at it, the reader should keep in mind that the same SEV approach can use more than one metric, and that we considered in this chart only the commonly used metrics, as described in Section 3.3.3.

Fig. 20 shows that *change comprehension*, *change prediction*, and *contribution analysis* are the most common applications of SEV tools. For that the researchers have heavily used *coupling (any)*, *number of authors* and *number of commits* metrics. *Feasibility study* is the main type of evaluation used by these three facets. *Change comprehension* heavily used *feasibility studies* and *case studies* or did *no* evaluation. While *change prediction* and *contribution analysis* mostly used *feasibility studies* or did *no* evaluation.

Fig. 20 is very helpful to identify gaps and possible research directions. Some gaps are large and easy to spot. The *comparisons with other tools* and *controlled experiments* types of evaluation, for example, have not been explored. Other gaps are subtler. We have only two *process improvement* studies in the mapping and they do not use any one of the main metrics used by other approaches. *Complexity* and *size* (LOC) have not being used in *re-documentation* studies. *Coupling*, *number of commits* and *number of authors* have not been used for *defect prediction*, and so on so forth.

3.4. The way SEV tools are used to execute software engineering tasks

To answer research questions (RQ) Q8 and Q9, this last section focuses on the interface between software engineering and visualization. Here we discuss the *perspectives* and *strategies* of analysis adopted by the SEV approaches, and relate those with their software engineering goals.

3.4.1. Perspectives (RQ Q8)

The perspectives are concerned with the way in which one looks at the software. The software can be visually examined through different perspectives [7]. In the context of software visualization goals, a perspective represents a set of coordinated views designed to represent a group of properties of the object of analysis. A perspective can also consider the professional in question, for example, the analyst's perspective, the perspective of the programmer, etc.

In this mapping, we extracted the perspectives used by the authors of SEV papers. Based on what we found, we classified the studies according to the following perspectives: structural, inheritance, dependency (module or logical coupling), dynamic execution traces, dynamic exception flows, changes, defects and issues, features, clones and authorship.

Fig. 21 summarizes the most common perspectives found by us. They are the Structural, Dependency, Change, Authorship and Inheritance perspectives.

The *Structural* perspective shows the structure of the software, like package, class, method, for OO systems, and file, functions, procedures, for non OO systems. Several visual metaphors have been used to display this type of information. Treemaps [20], that display software modules as a set of nested rectangles, are used in several studies [S39, S118, S125, S134]. City metaphors that show the structures of the software as buildings are used in an even larger set of the studies [S3, S6, S17, S63, S117].

The *Inheritance* perspective is concerned with OO inheritance. Lanza's Polymetric view [25] is the most famous view to address this perspective [S88, S118, S124, S125].

The *Dependency* perspective reveals the relationship among software modules. For that, it is common to use graphs, and there are many ways developed to draw them in a computer screen [16,33]. Some examples found in the mapping are [S31, S43, S58, S59, S60].

The *Authorship* perspective brings out the authors activities during software evolution, like how many commits they have performed, or which files they have worked on. It answers questions like "Who performed these modifications of the code?"

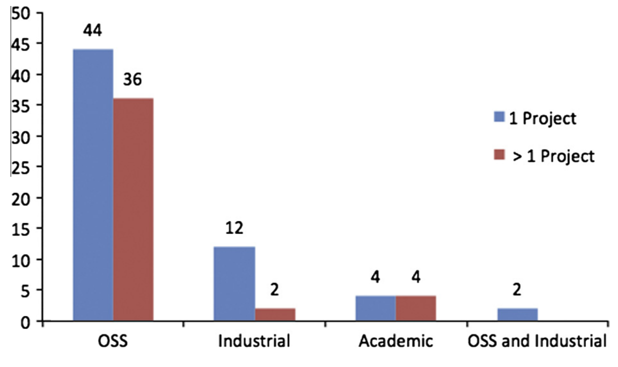

Fig. 19. Feasibility study per type and number of projects.

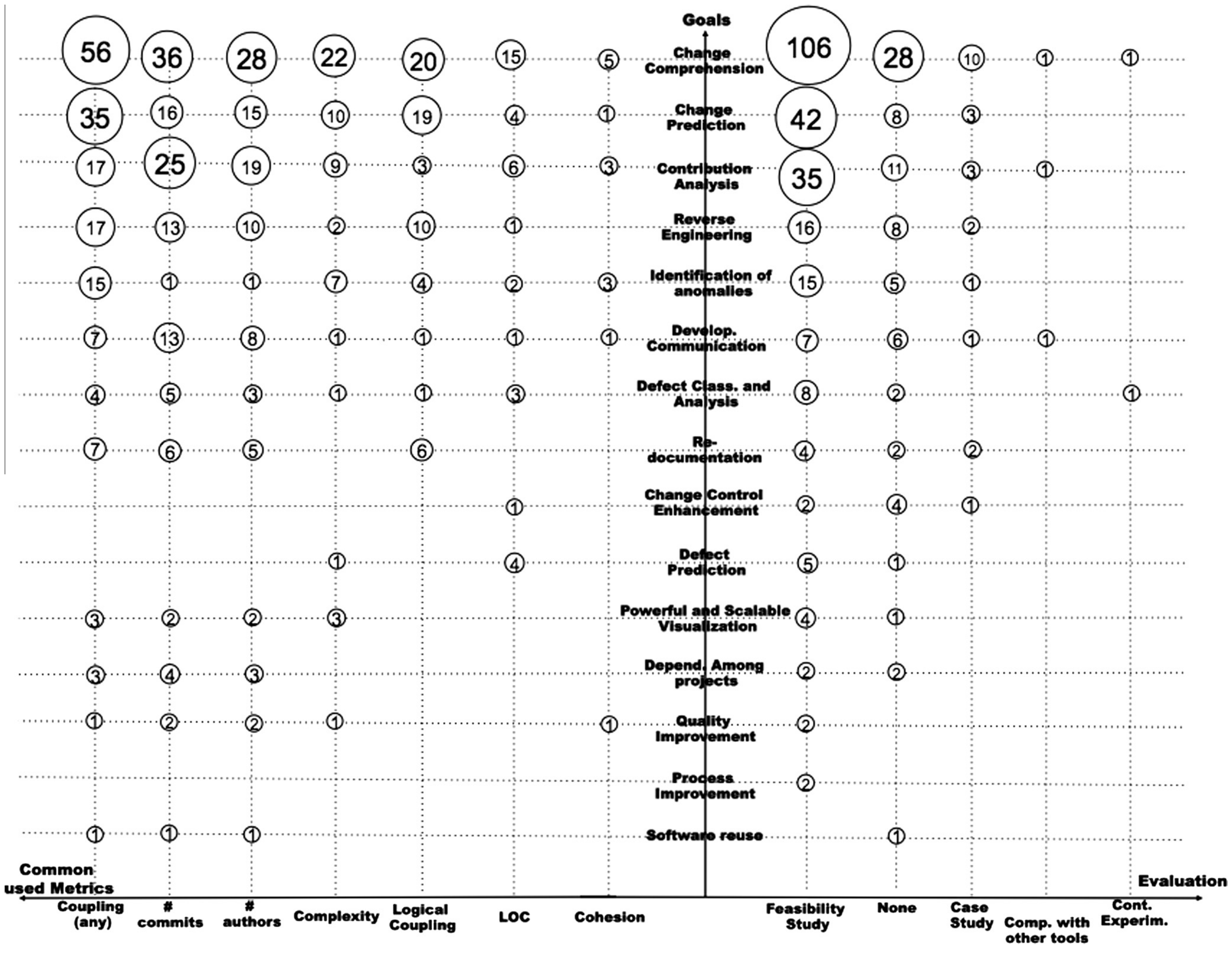

Fig. 20. Goals × commonly used metrics × evaluation.

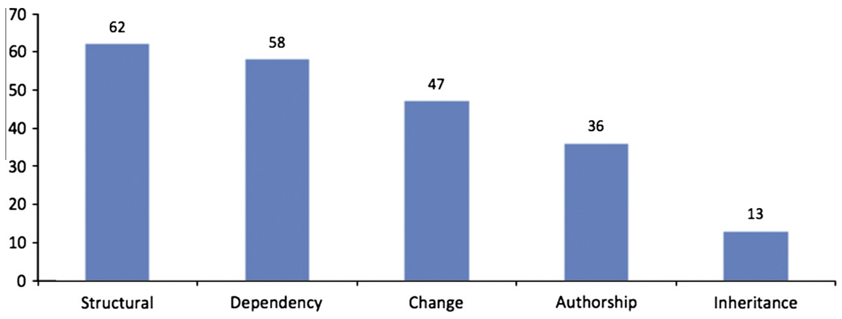

Fig. 21. Common perspectives found in the study.

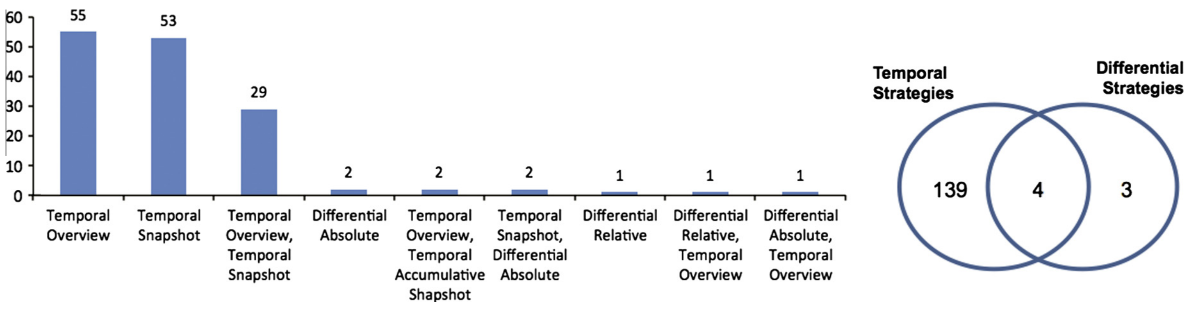

Fig. 22. Number of studies per strategy.

(e.g. [S103]). Many examples of the authorship perspective were found during our mapping study [S40, S75, S91, S109, S122, S141].

Lastly, the *Change* perspective uses change events as the SEV main object of analysis. It puts the changes in evidence for software evolution analysis [S104, S42].

It is not surprising that those are the most common perspectives. Structural, Inheritance and Dependency information are the most used in OO analysis, and Change and Author are properties intrinsically related to software evolution.

It is important to highlight that 80 studies (54.79%) use more than one perspective. It is recurrent the use of different views for each perspective (e.g. [S27]), however, it is also possible to use the same view with different perspectives (e.g. [S25, S26, S37]). A treemap, for example, can be used to visualize the modular structure or the inheritance structure of a software system.

3.4.2. Strategies (RQ Q9)

Software evolution can be visually analyzed using different strategies. Section 2.1.3 explains the analysis strategy classification scheme we adopted in our mapping study. The left side of Fig. 22 shows the number of studies we found per strategy or combinations of strategies.

According to our results, *Temporal Overview* is the most used strategy. It appears in 88 (60.27%) of the 146 primary studies. Of those, 55 (37.67%) studies use TO as its stand alone SEV analysis strategy. Likewise, *Temporal Snapshots* are used in 74 (50.68%) of the 146 primary studies. Of those, 53 (36.30%) studies use TS as its stand alone SEV analysis strategy. On top of that, 141 (96.58%) studies use at least one of these two strategies.

Moreover, only 35 (23.97%) studies use more than one strategy. *Temporal Overview* and *Temporal Snapshot* are the most common strategy combination, occurring in 29 (19.86%) of the studies. The *Differential Absolute* strategy, the most common in its class, is only used in five of the studies (two of them as the standalone strategy).

The right side of Fig. 22 shows a Venn diagram summarizing the use of temporal and differential strategies. It confirms that the differential strategies are much less used.

3.4.3. Goals × perspectives × strategies

Fig. 23 uses a bubble chart to display *goals versus perspectives* and *goals versus strategies*. It summarizes our main results showing trends and gaps among the analyzed studies. While looking at it, the reader should keep in mind that the same SEV approach can have more than one goal, use more than one perspective and apply more than one strategy to achieve its goals.

As discussed in Section 3.3.5, *Change Comprehension*, *Change Prediction*, and *Contribution Analysis* are the most used application of SEV tools. According to Fig. 23, one can observe that the researchers have heavily used *Temporal Overview* and *Temporal Snapshot* strategies. *Change Comprehension* heavily used *Structural*,

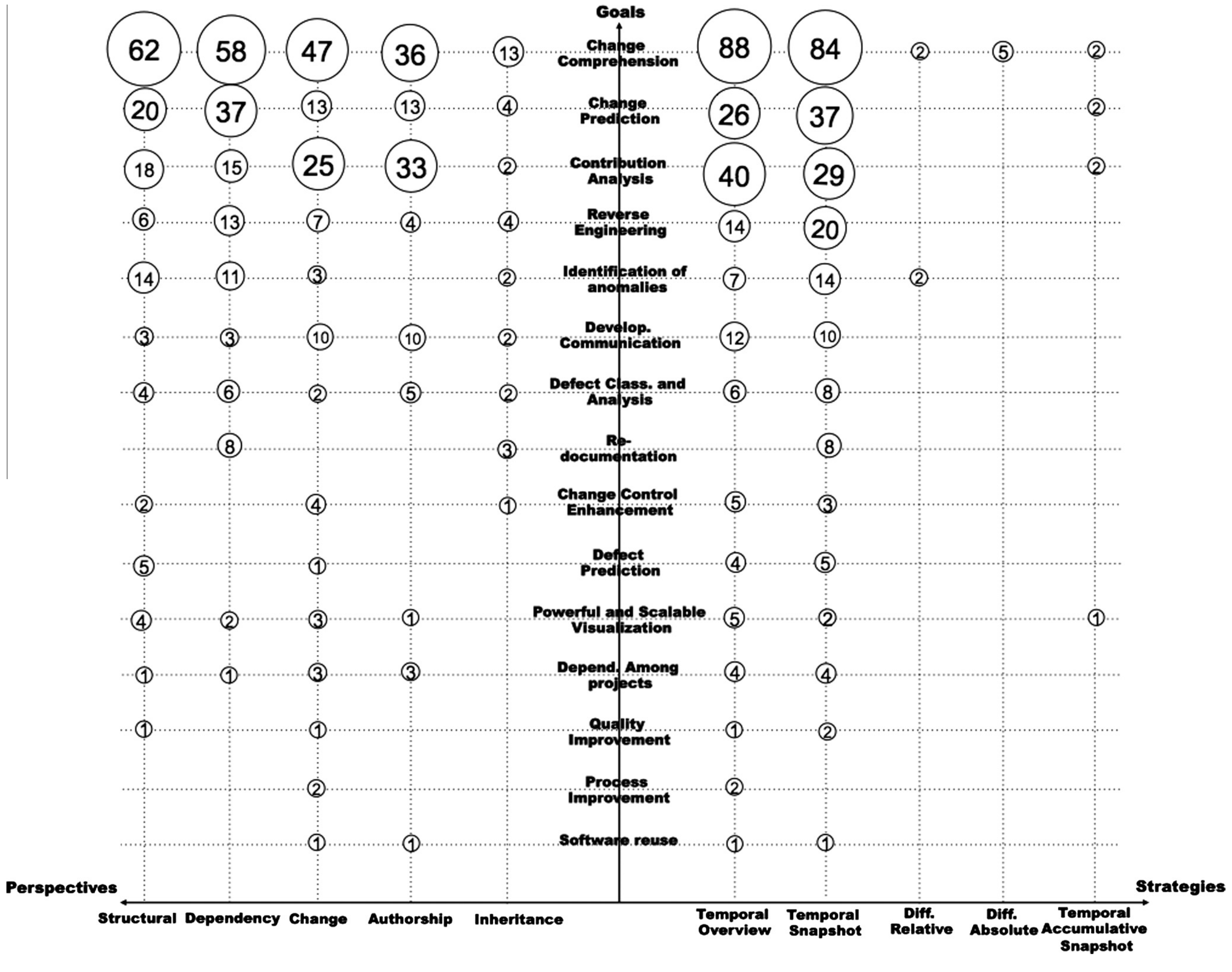

Fig. 23. Goals × perspectives × strategies.

Dependency, *Change* and *Authorship* perspectives, while *Change Prediction* used *Structural* and *Dependency* most and *Contribution Analysis* used *Change* and *Author* perspective most.

Fig. 23 is also very helpful to identify gaps and possible research direction. Some gaps are easy to spot. The differential approaches, for example, have clearly been underexplored. Other gaps are subtler. We have only six defect prediction studies in the mapping and almost all of them analyze the software only from the *Structural* perspective. However, there are software visualization studies that highlight the importance of the *Inheritance* perspective in the analysis of software defects [30]. There is no reason one should not adapt this perspective for *defect evolution analysis*.

3.5. Discussion

The number of published studies in SEV, according to this mapping study, is decreasing. This may indicate three things: (1) that the level of interest in the subject (software evolution) is decreasing; (2) that the area has already produced definitive results; or (3) that the area is failing to reach its goals. We believe that we are dealing with the third case. Our assumption is based on the fact that: (1) software evolution is a very important topic in modern software engineering; and (2) SEV is a broad and relatively young research area, that has much to offer to the software evolution research community. This section focuses on discussing the main findings, flaws and limitations of the SEV approaches we observed in this systematic mapping study.

3.5.1. Some SEV approaches do not always focus on concrete SE goals

In this mapping study, we tried to clearly capture the goals and the tasks addressed by the proposed SEV approaches. Unfortunately, several studies only describe their goal at a very high level of abstraction. We observed that some SEV approaches do not always focus on achieving practical software engineering goals, such as identification of candidates for refactoring [38], predict buggy modules [39,40], or feature modularization [41,42]. Some of them, for example, aim at producing *powerful and scalable visualizations* [S33][S104], without consideration for how those visualizations will be used to perform concrete software engineering tasks.

Visualization approaches should be task specific. It is necessary to specify which SE goals they aim to reach, and evaluate if they achieved those goals. When presenting their tools, many authors simply present examples of gained insights, without giving systematic guidelines of how these insights shall be used. SEV tools must be associated with usage methodologies and guidelines.

3.5.2. SEV approaches must be validated

Matters are not helped by the fact that most approaches are only partially validated, if validated at all, as the number of case studies and controlled experiments suggest (see Section 3.3.4). Any SEV approach must focus on the end user and validate its usefulness in well-executed experiments or case studies. This is far from proposing a new visually appealing visualization and executing a few feasibility studies over it. Researchers should focus on answering questions like: Does my approach scale to larger real world software systems? How my results generalize to other users, domains, and systems? Otherwise, there will be always an issue with the external validity of the proposed approach and, worse, the approach will have a tough time to move from the state of the art to the state of the practice in software engineering.

Unfortunately, the results of our mapping show that there is little robust validation in the area. The number of feasibility study is very high (106–72.60%) and these studies show the results based on the researcher point of view. We found only 10 case studies that were executed with subjects. On top of that, only one study applied a controlled experiment to validate its approach [S114], and it did not compare its visualization against another one. This absence of strong validation might be explained by the high cost and effort associated with these studies. Nonetheless, this validation effort would well be worthwhile, especially if associated with real software engineering goals, as previously discussed.

3.5.3. Lack of cooperative and comparative work

We also found out that there is actually little collaborative work in the area. Most of the analyzed work tries to develop new visualization approaches as oppose as to validate or add value to existing ones. Validation and cooperative activities would lead to faster improvement of existing approaches and to a deeper understanding of the area.

We observed that many feasibility studies used the same open source software, like Mozilla, Argo UML and Postgres. However, the authors never benchmarked their results against other approaches.

There is also a lack of use of a common terminology to classify the approaches from a software engineering point of view. The systematic use of a common terminology, such as the classification scheme used in Section 3.2 and 3.3, would go a long way to better organize the area for future secondary studies on the approaches' novelty, complementarities and usefulness.

Finally, this mapping study also revealed that the majority of the approaches use only one view. This indicates a lack of follow up work and collaboration in the area, as clearly many approaches would profit from combining and reusing well-established views.

3.5.4. Combining views and strategies

According to the mapping study results, Temporal Overview is the most used strategy to represent software evolution. Despite the contribution of Temporal Overview strategies, the visualization of particular software entities, in the way provided by the other strategies, is more valuable for many software engineering activities. As it has to represent a lot of data in each visual scene, we observed in this mapping that temporal overviews many times use pixel oriented visual paradigms (e.g. [S37, S42]). They are visually appealing, scalable, and provide a way to see patterns in the data, but summarize information and generally do not explicitly represent particular software entities. They make it difficult to identify what to do next and how to select particular software entities to a detailed investigation of their characteristics. Consider the approach described in [S74] as an example, it has an interesting overview strategy, but it does not provide mechanisms to analyze the software modules themselves.

The best approach in these cases is to combine global views with temporal overviews of just one (or few) module (e.g. [S15, S39, S28]). This way one has the opportunity to use the temporal views for selecting and examining specific modules on demand. Kiviat diagrams [S44] and Parallel Coordinates [S13] are examples of views that can be used for this purpose.

When present, the combination of strategies occurs in different ways. Some studies start with temporal overview views of the software and refine them focusing on the details of high-level entities such as packages, classes and methods [S90]. Others [S94] start with a temporal overview, and later detail using a temporal snapshot strategy. Or still apply the opposite scenario, starting by a temporal snapshot, followed by a temporal overview of a selected module of interest [S118]. These types of approaches conform better with the mantra "Overview first, zoom and filter, then details on demand" [32].

In spite of the approaches discussed above, the lack of empirical studies does not permit to evaluate what are the best ways to combine views and strategies for SEV. This is an important research issue for the area.

3.5.5. On the use of metrics and data sources

We also observed that there is much opportunity to combine data sources and metrics on SEV. Fig. 16 shows, for example, that information from BTS and source code were never combined on the universe of mapped primary studies. There are several software engineering tasks (e.g., identifying fault prone modules) that are better tackled by combining different data sources.

Although SEV approaches facilitate the usage of metrics, it is important to remark that this can be used for good and evil. On the good side, metrics are displayed in context, and are easy to explore and visualize. On the bad side, the limitations of some metrics [17], recurrently used in the studies we found in this mapping, may be hidden by attractive visualization paradigms. Although this is a subject from software measurement theory, it is also a very important issue for those who work with software visualization.

3.5.6. On our main research question

The main research question of this work is "**What maintenance tasks are current SEV technologies evidently supporting and how do the approaches differ from each other?**" As previously discussed, this question derived nine sub-questions and a set of facets related to software visualization, software engineering and the intersection between these two areas.

Our main results are summarized in the bubble charts of Figs. 20 and 23. In them, we observed that the SEV approaches are in fact basically used to address maintenance goals. Examples of the most attacked goals are: change comprehension, change prediction, contribution analysis, reverse engineering, identification of anomalies, and development communication. Despite of that, SEV has been used for other purposes such as process improvement.

Besides the maintenance tasks, the SEV approaches differ in many other dimensions. Fig. 16 shows that SEV approaches use mostly three types of data source, but source code is never used in combination with bug tracking system data. As a result, structural metrics such as LOC and complexity is not used in combination with metrics like number of defects. Fig. 23 also shows that SEV approaches differ in perspectives, but not so much in strategies of analysis, as most strategies are temporal (overview and snapshot).

4. Threats to validity

There are some threats to validity in our study. They are presented below with the strategies for its mitigation.

Research question: The research questions defined in this study may not provide complete coverage of the SEV area. The questions were defined based on two points of view: software visualization and software engineering. For software visualization, the questions address the main concepts of the area, as proposed in two taxonomies [27,32]. For SE, we followed a GQM-like approach, defining facets we believe are important for the area.

Search strings: Even though the search strings are broad, it is possible that they did not address some studies. We mitigated this threat by snow-balling the references of the primary studies to find other studies.

Other threat related to the search string is that, as it was too broad, it returned a large number of papers (8945). Looking at them required a large effort and the activity may have introduced fatigue in the process. We took two measures to reduce human error and bias during this activity. To reduce fatigue, and consequently human error, each review section lasted at most 4 h. To reduce bias, this step, as well as the others, was performed by two researchers, and when both disagreed on applying the inclusion or exclusion criteria, we considered a third opinion.

Publication bias: We cannot guarantee that all relevant primary studies were selected. Even though we searched the main digital libraries in our field, it is possible that some relevant studies were not included in the search process. Like before, we mitigated this threat by snow-balling the references of the primary studies to find other studies.

Search conducted: The search processes in the digital databases are not exactly the same, and we do not know how they work internally. We mitigate this treat by adapting the search strings for each digital database and assumed that equivalent logical expressions worked consistently across all the databases.

Data extraction: During the extraction process, the studies were classified based on our judgment. This means that some studies could have been classified incorrectly. In order to mitigate this threat, the classification process was performed always by two researchers, with a third researcher involved in a discussion process in case of conflict. Another threat is that the three data collection and classification researchers came from the same organization. This may have biased the interpretation of some of the analyzed concepts. We did not mitigate this threat.

5. Concluding remarks and future work

The purpose of this work was to perform a systematic mapping study that helps to structure the field of research (software evolution visualization) and identify the relevant literature discussing visualization in software evolution through the main question: "**What maintenance tasks are current SEV technologies evidently supporting and how do the approaches differ from each other?**"

As the software evolution area is complex, we observed that there is a necessity for multi-metrics, multi-perspective, multi-strategy SEV, to address the many existing software engineering goals.

In software visualization research, it is almost always necessary to implement a tool for each proposed visualization approach. However, the authors present their tools in very diverse manners. It is necessary to standardize at least some of the issues that should be discussed in this type of paper. The information visualization (InfoVis) and the empirical software engineering (ESE) fields may provide some of these key issues. SEV papers should always discuss interaction mechanisms like an InfoVis paper. They should also discuss their approach goals and validation strategy like an ESE paper. The adoption of taxonomies could also help to standardize the area. As an example, [S64] and [S144] already used Maletic's taxonomy [27] to classify their work.

One of the critical results found in this study is that there is little formal validation and collaboration in the area. Careful evaluation of the proposed approaches would increase the confidence on its results. Collaboration and reuse of results would speed up development and improve the maturity of those approaches.

Acknowledgments

This work has received full or partial funding from the following agencies and projects: for the authors Renato and Manoel PROCAD-

Table 7
Digital Databases.

ACM Digital Library	http://www.portal.acm.org/dl.cfm
Elsevier – Engineering Village – Compendex	http://www.engineeringvillage.com
IEEEXplore	http://www.ieeexplore.ieee.org
Scopus	http://www.scopus.com
SciVerse ScienceDirect	http://www.sciencedirect.com/
DPLP	http://www.dblp.org/search/

Table 8
Search string per digital databases.

Search Engines	Search strings
ACM Digital Library (Title, Abstract)	(Abstract:(software OR system) AND Abstract:(evolution OR evolving OR evolve) AND Abstract:(visualization OR visual OR visualisation)) OR (Title:(software OR system) AND Title:(evolution OR evolving OR evolve) AND Title:(visual OR visualization OR visualisation))
Elsevier – Engineering Village (Subject, Title, Abstract)	("software" OR "system") AND ("visualization" OR "visual" OR "Visualisation") AND ("evolution" OR "evolving" OR "evolve")
IEEEXplore Metadata	("software" OR "system") AND ("visualization" OR "visual" OR "Visualisation") AND ("evolution" OR "evolving" OR "evolve")
Scopus	("software" OR "system") AND ("visualization" OR "visual" OR "Visualisation") AND ("evolution" OR "evolving" OR "evolve") TITLE-ABS-KEY(("software" OR "system") AND ("visualization" OR "visual" OR "Visualisation") AND ("evolution" OR "evolving"))
SciVerse ScienceDirect	TITLE-ABS-KEY(("software" OR "system") AND ("visualization" OR "visual" OR "Visualisation") AND ("evolution" OR "evolving" OR "evolve"))
DBLP	Software|system evolution|evolving|evolve visualization|visual|visualisation

NF (Grant 720/2010) and INES (www.ines.org.br) funded by CNPq (Grant 573964/2008-4).

Appendix A. Search Engines

The search process will be performed in web indexed digital libraries (automatic search). The source list for digital databases is presented in Table 7.

Search strings for each Search Engine are presented in Table 8.

Appendix B. List of primary studies

[S1] R.K. Saha, C.K. Roy, K.A. Schneider, Visualizing the evolution of code clones, in: IWSC'11: Proceedings of the 5th International Workshop on Software Clones, ACM, New York, NY, USA, 2011, pp. 71–72.

[S2] C. Müller, G. Reina, M. Burch, D. Weiskopf, Subversion statistics sifter, in: ISVC'10: Proceedings of the 6th International Conference on Advances in Visual Computing – Volume Part III, Springer-Verlag, Heidelberg, Berlin, 2010, pp. 447–457.

[S3] F. Steinbruckner, C. Lewerentz, Representing development history in software cities, in: SOFTVIS '10: Proceedings of the 5th International Symposium on Software Visualization, ACM, New York, NY, USA, 2010, pp. 193–202.

[S4] M. Ogawa, k. Ma, Software evolution storylines, in: SOFTVIS '10: Proceedings of the 5th International Symposium on Software Visualization, ACM, New York, NY, USA, 2010, pp. 35–42.

[S5] M. Goeminne, T. Mens, A framework for analysing and visualising open source software ecosystems, in: IWPSE-EVOL'10: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), New York, NY, USA, 2010, p. 47.

[S6] F. Steinbruckner, Coherent software cities, in: ICSM'10: Proceedings of the IEEE International Conference on Software Maintenance, IEEE Computer Society, Washington, DC, USA, 2010, pp. 1–2.

[S7] M. Lungu, M. Lanza, The small project observatory: a tool for reverse engineering software ecosystems, in: ICSE'10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering – Volume 2, ACM, New York, NY, USA, 2010, pp. 289–292.

[S8] M. D'Ambros, M. Lanza, R. Robbes, Commit 2.0, in: Web2SE'10: Proceedings of the 1st Workshop on Web 2.0 for Software Engineering, ACM, New York, NY, USA, 2010, pp. 14–19.

[S9] Kuhn, D. Erni, P. Loretan, O. Nierstrasz, Software cartography: thematic software visualization with consistent layout, Journal of Software Maintenance 22 (3) (2010) 191–210.

[S10] X. Wu, A. Murray, M. Storey, R. Lintern, A reverse engineering approach to support software maintenance: version control knowledge extraction, In Reverse Engineering, 2004. Proceedings. 11th Working Conference, vol., No., pp.90,99, 8–12 Nov. 2004.

[S11] M. D'Ambros, M. Lanza, M. Lungu, Visualizing co-change information with the evolution radar. IEEE Transactions on Software Engineering, 35 (5) (2009) 720–735.

[S12] K. Wnuk, B. Regnell, L. Karlsson, What happened to our features? visualization and understanding of scope change dynamics in a large-scale industrial setting, in: RE'09: Proceedings of the 17th IEEE International Requirements Engineering Conference, IEEE Computer Society, Washington, DC, USA, 2009, pp. 89–98.

[S13] Gonzalez, R. Theron, A. Telea, F.J. Garcia, Combined visualization of structural and metric information for software evolution analysis, in: IWPSE-Evol'09: Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops, ACM, New York, NY, USA, 2009, pp. 25–30.

[S14] J. Ekanayake, J. Tappolet, H.C. Gall, A. Bernstein, Tracking concept drift of software projects using defect prediction quality, in: MSR'09: Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories, IEEE Computer Society, Washington, DC, USA, 2009. pp. 51–60.

[S15] Jermakovics, R. Moser, A. Sillitti, G. Succi, Visualizing software evolution with lagrein, in: OOPSLA'08: Companion to the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications, ACM, New York, NY, USA, 2008, pp. 749–750.

[S16] Kuhn, P. Loretan, O. Nierstrasz, Consistent layout for thematic software maps, in: WCRE'08: Proceedings of the 15th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2008, pp. 209–218.

[S17] R. Wettel, M. Lanza, Visual exploration of large-scale system evolution, in: WCRE'08: Proceedings of the 15th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2008, pp. 219–228.

[S18] R. Theron, A. Gonzalez, F. J. Garcia, Supporting the understanding of the evolution of software items, in: SoftVis'08: Proceedings of the 4th ACM Symposium on Software Visualization, ACM, New York, NY, USA, 2008, pp. 189–192.

[S19] G. Langelier, H. Sahraoui, P. Poulin, Exploring the evolution of software quality with animated visualization, in: VLHCC'08: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, IEEE Computer Society, Washington, DC, USA, 2008, pp. 13–20.

[S20] M. D'Ambros, M. Lanza, A flexible framework to support collaborative software evolution analysis, in: CSMR'08: Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2008, pp. 3–12.

[S21] N. Hanakawa, Visualization for software evolution based on logical coupling and module coupling, in: APSEC'07: Proceedings of the 14th Asia–Pacific Software Engineering Conference, IEEE Computer Society, Washington, DC, USA, 2007, pp. 214–221.

[S22] R. Therón, A. Gonzalez, F.J. García, P. Santos, The use of information visualization to support software configuration management, in: INTERACT'07: Proceedings of the 11th IFIP TC 13 International conference on Human–computer interaction – Volume Part II, Springer-Verlag, Heidelberg, Berlin, 2007, pp. 317–331.

[S23] Jermakovics, M. Scotto, G. Succi, Visual identification of software evolution patterns, in: IWPSE'07: Ninth International Workshop on Principles of software evolution: in Conjunction With The 6th ESEC/FSE Joint Meeting, ACM, New York, NY, USA, 2007, pp. 27–30.

[S24] M. Lungu, T. Girba, A small observatory for super-repositories, in: IWPSE'07: Ninth International Workshop on Principles of Software Evolution: in Conjunction with the 6th ESEC/FSE Joint Meeting, ACM, New York, NY, USA, 2007, pp. 106–109.

[S25] L. Voinea, A. Telea, Visual analytics: visual data mining and analysis of software repositories, Computers and Graphics 31 (3) (2007) 410–428.

[S26] L. Voinea, Johan Lukkien, A. Telea, Visual assessment of software Evolution, Science of Computer Programming 65 (3) (2007) 222–248.

[S27] M. D'Ambros, M. Lanza, Bugcrawler: visualizing evolving software systems, in: CSMR'07: Proceedings of the 11th European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2007, pp. 333–334.

[S28] M. Lungu, M. Lanza. Exploring inter-module relationships in evolving software systems, in: CSMR'07: Proceedings of the 11th European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2007, pp. 91–102.

[S29] D. Beyer, A. E. Hassan, Animated visualization of software history using evolution storyboards, in: WCRE'06: Proceedings of the 13th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2006, pp. 199–210.

[S30] M. D'Ambros, M. Lanza, Reverse engineering with logical coupling, in: WCRE'06: Proceedings of the 13th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2006, pp. 189–198.

[S31] M. Fischer, H. Gall, Evograph: a lightweight approach to evolutionary and structural analysis of large software systems, in: WCRE'06: Proceedings of the 13th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2006, pp. 179–188.

[S32] Xi. Xie, D. Poshyvanyk, A. Marcus. Visualization of CVS repository information, in: WCRE'06: Proceedings of the 13th Working Conferenceon Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2006, pp. 231–242.

[S33] L. Voinea, A. Telea, Multiscale and multivariate visualizations of software evolution, in: SoftVis'06: Proceedings of the 2006 ACM Symposium on Software Visualization, ACM, New York, NY, USA, 2006, pp. 115–124.

[S34] M. Balint, R. Marinescu, T. Gîrba, How developers copy, in: ICPC'06: Proceedings of the 14th IEEE International Conference on Program Comprehension, IEEE Computer Society, Washington, DC, USA, 2006, pp. 56–68.

[S35] D. Beyer, A. E. Hassan, Evolution storyboards: visualization of software structure dynamics, in: ICPC'06: Proceedings of the 14th IEEE International Conference on Program Comprehension, IEEE Computer Society Washington, DC, USA, 2006, pp. 248–251.

[S36] M. D'Ambros, M. Lanza, M. Lungu, The evolution radar: visualizing integrated logical coupling information, in: MSR'06: Proceedings of the 2006 International Workshop on Mining Software Repositories, ACM, New York, NY, USA, 2006, pp. 26–32.

[S37] L. Voinea, A. Telea, An open framework for cvs repository querying, analysis and visualization, in: MSR'06: Proceedings of the 2006 International Workshop on Mining Software Repositories, ACM, New York, NY, USA, 2006, pp. 33–39.

[S38] M. D'Ambros, M. Lanza, Software bugs and evolution: a visual approach to uncover their relationship, in: CSMR'06: Proceedings of the Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2006, pp. 229–238.

[S39] G. Lommerse, F. Nossin, L. Voinea, A. Telea, The visual code navigator: an interactive toolset for source code investigation, in: INFOVIS'05: Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization, IEEE Computer Society, Washington, DC, USA, 2005, p. 4.

[S40] M. D'Ambros, M. Lanza, H. Gall, Fractal figures: Visualizing development effort for CVS entities, in: VISSOFT'05: Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, IEEE Computer Society, Washington, DC, USA, 2005, p. 16.

[S41] Mesnage, M. Lanza, White Coats: Web-visualization of evolving software in 3D, in: VISSOFT'05: Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, IEEE Computer Society, Washington, DC, USA, 2005, p. 15.

[S42] A.E. Hassan, J. Wu, R.C. Holt, Visualizing historical data using spectrographs, in: METRICS'05: Proceedings of the 11th IEEE International Software Metrics Symposium, IEEE Computer Society, Washington, DC, USA, 2005, p. 31.

[S43] J. Ratzinger, M. Fischer, H. Gall, Evolens: lens-view visualizations of evolution data, in: IWPSE'05: Proceedings of the Eighth International Workshop on Principles of Software Evolution, IEEE Computer Society, Washington, DC, USA, 2005, pp. 103–112.

[S44] M. Fischer, J. Oberleitner, H. Gall, T. Gschwind, System evolution tracking through execution trace analysis, in: IWPC'05: Proceedings of the 13th International Workshop on Program Comprehension, IEEE Computer Society, Washington, DC, USA, 2005. pp. 237–246.

[S45] M. Pinzger, H. Gall, M. Fischer, M. Lanza, Visualizing multiple evolution metrics, in: SoftVis'05: Proceedings of the 2005 ACM Symposium on Software Visualization, ACM, New York, NY, USA, 2005, pp. 67–75.

[S46] J. Wu, R.C. Holt, A. E. Hassan, Exploring software evolution using spectrographs, in: WCRE'04: Proceedings of the 11th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2004, pp. 80–89.

[S47] M. Fischer, H. Gall, Visualizing feature evolution of large-scale software based on problem and modification report data: Research article, in: Journal of Software Maintenance and Evolution: Research and Practice – Analyzing the Evolution of Large-Scale Software 16 (6) (2004) 385–403.

[S48] F.V. Rysselberghe, S. Demeyer, Studying software evolution information by visualizing the change history, in: ICSM'04: Proceedings of the 20th IEEE International Conference on Software Maintenance, IEEE Computer Society, Washington, DC, USA, 2004, pp. 328–337.

[S49] J. Wu, C.W. Spitzer, A.E. Hassan, R.C. Holt, Evolution spectrographs: visualizing punctuated change in software evolution, in: IWPSE'04: Proceedings of the Principles of Software Evolution, 7th International Workshop, IEEE Computer Society Washington, DC, USA, 2004, pp. 57–66.

[S50] M. Fischer, M. Pinzger, H. Gall, Analyzing and relating bug report data for feature tracking, in: WCRE'03: Proceedings of the 10th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2003, p. 90.

[S51] Collberg, S. Kobourov, J. Nagra, J. Pitts, K. Wampler, A system for graph-based visualization of the evolution of software, in: SoftVis'03: Proceedings of the 2003 ACM Symposium on Software Visualization, ACM, New York, NY, USA, 2003, p. 77.

[S52] Q. Tu, M.W. Godfrey, An integrated approach for studying architectural evolution, in: IWPC'02: Proceedings of the 10th International Workshop on Program Comprehension, IEEE Computer Society Washington, DC, USA, 2002, p. 127.

[S53] M. Godfrey, Q. Tu, Growth, Evolution, and structural change in open source software, in: IWPSE'01:"Proceedings of the 4th International Workshop on Principles of Software Evolution, ACM, New York, NY, USA, 2001, pp. 103–106.

[S54] M. Lanza, The evolution matrix: recovering software evolution using software visualization techniques, in: IWPSE'01: Proceedings of the 4th International Workshop on Principles of Software Evolution, ACM, New York, NY, USA, 2001, pp. 37–42.

[S55] Knight, M. Munro, Organisational trails through software systems, in: IWPSE'01: Proceedings of the 4th International Workshop on Principles of Software Evolution, ACM, New York, NY, USA, 2001, pp. 150–153.

[S56] C. Riva, Visualizing software release histories with 3DSoftvis, in: ICSE'00: Proceedings of the 22nd International Conference on Software Engineering, ACM, New York, NY, USA, 2000, pp. 789.

[S57] H. Gall, M. Jazayeri, C. Riva, Visualizing software release histories: the use of color and third dimension, in: ICSM'99: Proceedings of the IEEE International Conference on Software Maintenance, IEEE Computer Society, Washington, DC, USA, 1999, p. 99.

[S58] R. Holt, J.Y. Pak, Gase: visualizing software evolution-in-the-large, in: WCRE'96: Proceedings of the 3rd Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 1996, p. 163.

[S59] Vanya, R. Premraj, H. van Vliet, Interactive exploration of co-evolving software entities, in: CSMR'10: Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2010, pp. 260–263.

[S60] L. Schrettner, P. Hegedus, R. Ferenc, L.J. Fulop, T. Bakota, Development of a methodology, software-suite and service for supporting software architecture reconstruction, in: CSMR'10: Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2010, pp. 190–193.

[S61] S. W. Thomas, B. Adams, A.E. Hassan, D. Blostein, Validating the use of topic models for software evolution, in: SCAM '10: Proceedings of the 2010 10th IEEE Working Conference on Source Code Analysis and Manipulation, IEEE Computer Society, Washington, DC, USA, 2010, pp. 55–64.

[S62] M. Ogawa, K. Ma, Code swarm: a design study in organic software visualization, IEEE Transactions on Visualization and Computer Graphics. November, 2009.

[S63] R. Wettel, Visual exploration of large-scale evolving software, in: ICSE'09: 31st International Conference on Software Engineering, Companion Volume. IEEE, Vancouver, Canada, May 16–24, 2009, pp. 391–394.

[S64] C. Treude, M. Storey, Concernlines: a timeline view of co-occurring concerns, in: ICSE'09: Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, Washington, DC, USA, 2009, pp. 575–578.

[S65] G. Robles, J.M. Gonzalez-Barahona, I. Herraiz, Evolution of the core team of developers in libre software projects, in: MSR'09: Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, IEEE Computer Society, Washington, DC, USA, 2009, pp. 167–170.

[S66] Hindle, M.W. Godfrey, R.C. Holt, What's hot and what's not: windowed developer topic analysis, in: ICSM'09: IEEE International Conference on Software Maintenance, IEEE, Waterloo, Canada 2009, pp. 339–348.

[S67] S. Wenzel, J. Koch, U. Kelter, A. Kolb, Evolution analysis with animated and 3D-visualizations, in: ICSM'09: IEEE International Conference on Software Maintenance, Waterloo, Canada 2009, pp. 475–478.

[S68] K. Wnuk, B. Regnell, L. Karlsson, Feature transition charts for visualization of cross-project scope evolution in large-scale requirements engineering for product lines, Fourth International Workshop on Requirements Engineering Visualization, DC, USA, 2009, pp. 11–20.

[S69] R. Robbes, M. Lanza, Spyware: A change-aware development toolset, in: ICSE'08: Proceedings of the 30th International Conference on Software engineering, ACM, New York, NY, USA, 2008, pp. 847–850.

[S70] Y. Yu, M. Wermelinger, Graph-centric tools for understanding the evolution and relationships of software structures, in: WCRE'08: Proceedings of the 2008 15th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2008. pp. 329–330.

[S71] M. D'Ambros, Supporting software evolution analysis with historical dependencies and defect information, in: ICSM'08: International Conference on Software Maintenance, IEEE, 2008, pp. 412–415.

[S72] S. Boccuzzo, H. Gall, Software visualization with audio supported cognitive glyphs, in: ICSM'08: International Conference on Software Maintenance, IEEE, 2008, pp. 366–375.

[S73] M. Lungu, Towards reverse engineering software ecosystems, in: ICSM'08: International Conference on Software Maintenance, IEEE, 2008, pp. 428–431.

[S74] T. Arbuckle, Visually summarising software change, in: IV'08: Proceedings of the 2008 12th International Conference Information Visualisation, IEEE Computer Society, Washington, DC, USA, 2008, pp. 559–568.

[S75] R.M. Ripley, A. Sarma, A. van der Hoek, A visualization for software project awareness and evolution, in: VISSOFT'07: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007, pp. 137–144.

[S76] L. Voinea, A. Telea, Visualizing debugging activity in source code repositories, in: VISSOFT'07: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007, pp. 156–157.

[S77] C.F.J. Lange, M.A.M. Wijns, M.R.V. Chaudron, MetricView-Evolution: UML-based views for monitoring model evolution and quality, in: CSMR'07: Proceedings of the 11th European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2007, pp. 327–328.

[S78] S.A. Bohner, D. Gracanin, T. Henry, K. Matkovic, Evolutional insights from UML and source code versions using information visualization and visual analysis, in: VISSOFT'07: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007, pp. 145–148.
[S79] Adar, M. Kim, SoftGUESS: visualization and exploration of code clones in context, in: ICSE'07: Proceedings of the 29th International Conference on Software Engineering, Washington, IEEE Computer Society, DC, USA, 2007, pp. 762–766.
[S80] Hindle, Z.M. Jiang, W. Koleilat, M.W. Godfrey, R.C. Holt, YARN: Animating Software Evolution, in: VISSOFT'07: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007, pp. 129–136.
[S81] Langelier, K.Dhambri, Visual analysis of Azureus using verso, in: VISSOFT'07: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007, pp. 163–164.
[S82] M. D'Ambros, M. Lanza, M. Pinzger, "A Bug's Life" Visualizing a bug database, in: VISSOFT'07: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007 pp. 113–120
[S83] M. Ogawa, K.-L. Ma, C. Bird, P. Devanbu, A. Gourley, Visualizing social interaction in open source software projects, in: APVIS'07: 6th International Asia–Pacific Symposium on Visualization, 2007, pp. 25–32.
[S84] McNair, D. M. German, Jens Weber-Jahnke, Visualizing software architecture evolution using change-sets, in: WCRE'07: Proceedings of the 14th Working Conference on Reverse Engineering, IEEE Computer Society, Washington, DC, USA, 2007, pp. 130–139.
[S85] T.N. Nguyen, A novel structure-oriented difference approach for software artifacts, in: COMPSAC'06: Proceedings of the 30th Annual International Computer Software and Applications Conference – Volume 01, IEEE Computer Society, Washington, DC, USA, 2006, pp. 197–204.
[S86] Van Rysselberghe, Reconstructing higher level change information from versioning data, in: CSMR'06: Proceedings of the Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2006, pp. 331–333.
[S87] L. Voinea, A. Telea, Visual assessment techniques for component-based framework evolution, in: EUROMICRO'05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE Computer Society, Washington, DC, USA, 2005, pp. 168–179.
[S88] T. Gîrba, M. Lanza, S. Ducasse, Characterizing the evolution of class hierarchies, in: CSMR'05: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2005, pp. 2–11.
[S89] Telea, L. Voinea, Interactive visual mechanisms for exploring source code evolution, in: Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis. IEEE Press, 2005, pp. 52–57.
[S90] M. Burch, S. Diehl, P. Weibgerber, EPOsee – A tool for visualizing software evolution, in: VISSOFT'05: Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, IEEE Computer Society, Washington, DC, USA, 2005, p. 35.
[S91] T. Girba, A. Kuhn, M. Seeberger, S. Ducasse, How developers drive software evolution, in: IWPSE'05: Proceedings of the Eighth International Workshop on Principles of Software Evolution, IEEE Computer Society, Washington, DC, USA, 2005, pp. 113–122.
[S92] T. Zimmermann, S. Diehl, A. Zeller, How history justifies system architecture (or not), in: IWPSE'03: Proceedings of the 6th International Workshop on Principles of Software Evolution, IEEE Computer Society, Washington, DC, USA, 2003, p.73.
[S93] J.M. Bieman, A.A. Andrews, H.J. Yang, Understanding change-proneness in OO software through visualization, in: IWPC'03: Proceedings of the 11th IEEE International Workshop on Program Comprehension, IEEE Computer Society, Washington, DC, USA, 2003, p. 44.
[S94] Antoniol, G. Canfora, A. De Lucia, Maintaining traceability during object-oriented software evolution: a case study, in: ICSM'99: Proceedings IEEE International Conference on Software Maintenance, 1999, pp. 211–219.
[S95] D. Abramson, R. Sosic, A debugging tool for software evolution, in: Proceedings of the Seventh International Workshop on Computer-Aided Software Engineering, 1995, pp. 206–214.
[S96] M.J. Baker and S.G. Eick, Visualizing software systems, in: ICSE'94: Proceedings of the 16th International Conference on Software Engineering, Los Alamitos, IEEE Computer Society Press, CA, USA, 1994, pp. 59–67.
[S97] B. Gulla, Improved maintenance support by multi-version visualizations, in: Proceedings Conference on Software Maintenance, 1992, pp. 376–383.
[S98] M. D'Ambros, M Lanza, Visual software evolution reconstruction, Journal of Software Maintenance and Evolution 21(3) (2009) 217–232.
[S99] Y. Lee, J. Yang, Visualization of software evolution, in: Softwazre Engineering Research and Practice, 2008, pp. 343–348.
[S100] D. German, Abram Hindle, N. Jordan, Visualizing the evolution of software using softChange, in: SEKE'04: Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering, ACM Press, New York, NY, 2004, pp. 336–341.
[S101] D. German, A. Hindle, N. Jordan, Visualizing the evolution of software using softChange. International Journal of Software Engineering and Knowledge Engineering (IJSEKE), 2006.
[S102] M. Lanza, S. Ducasse, Understanding software evolution using a combination of software visualization and Software metrics, in: LMO'02: Proceedings of Languages et Modles Objets, Hermes Publications, 2002, pp. 135–149.
[S103] L. Voinea, A. Telea, J. J. van Wijk, CVSscan: visualization of code evolution, in: SoftVis'05: Proceedings of the 2005 ACM Symposium on Software Visualization, ACM, New York, NY, USA, 2005, pp. 47–56.
[S104] F. Chevalier, D. Auber, A. Telea, Structural analysis and visualization of C++ code evolution using syntax trees, in: IWPSE'07: Ninth International Workshop on Principles of Software Evolution: in conjunction with the 6th ESEC/FSE joint meeting, ACM, New York, NY, USA, 2007, pp. 90–97.
[S105] N. Zazworka, C. Ackermann, CodeVizard: A tool to aid the analysis of software evolution, in: ESEM'10: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, New York, NY, USA, 2010. p. 63.
[S106] Telea, D. Auber, Code Flows: Visualizing structural evolution of source code. Computer Graphics Forum 27 (3) (2008) 831–838, 2008.
[S107] L. Voinea, A. Telea, Mining software repositories with CVSGrab, in: MSR'06: Proceedings of the 2006 International Workshop on Mining Software Repositories, ACM, New York, NY, USA, 2006, pp. 167–168.

[S108] M. Lungu, M. Lanza, T. Gîrba, R. Robbes, The small project observatory: visualizing software ecosystems, in: WASDeTT'08: Workshop on Academic Software Development Tools and Techniques, 2008.

[S109] M. Lungu, M. Lanza, T. Gîrba, R. Robbes, The small project observatory: visualizing software ecosystems, Science of Computer Programming 75 (2010) 264–275.

[S110] M. D'Ambros, M, Lanza, Churrasco: Supporting collaborative software evolution analysis, in: WASDeTT'08: Workshop on Academic Software Development Tools and Techniques, 2008.

[S111] M. D'Ambros, M. Lanza, Distributed and collaborative software evolution analysis with Churrasco, Science of Computer Programming 75 (4) (2010) 276–287.

[S112] M.J. Baker, S.G. Eick, Space-filling software visualization, Journal of Visual Languages & Computing, 1995, pp. 160–182.

[S113] Cicchetti, D. Di Ruscio, A. Pierantonio, Model patches in model-driven engineering, in: MODELS'09: Proceedings of the 2009 International Conference on Models, in: Software Engineering, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 190–204.

[S114] C.E.A. Cunha, Y.C. Cavalcanti, P.A.M. Silveira Neto, E.S. Almeida, S.R.L. Meira, A visual bug report analysis and search tool, in: 22nd International Conference on Software Engineering and Knowledge Engineering, 2010.

[S115] R.S.V. Cepda, A.M. Magdaleno, L.G.P. Murta, C.M.L. Werner, Evoltrack: Improving design evolution awareness in Software development. Journal of the Brazilian Computer Society, 16 (2) (2010) 117–131.

[S116] D.M. German, An empirical study of fine-grained software modifications, in: ICSM'04: Proceedings of the 20th IEEE International Conference on Software Maintenance, IEEE Computer Society, Washington, DC, USA, 2004, pp. 316–325.

[S117] S.G. Eick, T.L. Graves, A.F. Karr, A. Mockus, P. Schuster, Visualizing Software changes. Transactions on Software Engineering 28 (4) (2002) 396–412.

[S118] R.L. Novais, C.A.N. Lima, G.F. Carneiro, R.M.S. Paulo, M. Mendonça, An interactive differential and temporal approach to visually analyze software evolution, in: VISSOFT'11: 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2011, pp. 1–4.

[S119] Garcia, N. Cacho, eFlowMining: An exception-flow analysis tool for .NET applications, in: LADCW'11: Fifth Latin-American Symposium on Dependable Computing Workshops, 2011, pp. 1–8.

[S120] G. Torres, R. Therón, F.J.G. Peñalvo, M. Wermelinger, Y. Yu, Maleku: An evolutionary visual software analysis tool for providing insights into software evolution, in: ICSM: 27th IEEE International Conference on Software Maintenance, 2011 pp. 594–597.

[S121] M. Burch, C. Vehlow, F. Beck, S. Diehl, D. Weiskopf, Parallel edge splatting for scalable dynamic graph visualization. IEEE Transactions on Visualization and Computer Graphics 17 (12) (2011) 2344–2353.

[S122] S. Neu, M. Lanza, L. Hattori, M. D'Ambros, Telling stories about gnome with complicity, in: VISSOFT'11: 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis, IEEE, 2011, pp. 1–8.

[S123] T. Omori, K. Maruyama, An editing-operation replayer with highlights supporting investigation of program modifications, in: IWPSE-EVOL'11: 12th International Workshop on Principles on Software Evolution and 7th ERCIM Workshop on Software Evolution, ACM, 2011, pp. 101–105.

[S124] Bergel, F. Baados, R. Robbes, W. Binder, Execution profiling blueprints. In: Software: Practice and Experience, first published online in Sep 15, 2011.

[S125] R.L. Novais, G.F. Carneiro, P.R.M. Simões Júnior, M.G. Mendonça, On the use of software visualization to analyze software evolution – an interactive differential approach, in: ICEIS'11: Proceedings of the 13th International Conference on Enterprise Information Systems, Beijing, 2011, pp. 15–24.

[S126] S.L. Voinea, A. Telea, A file based visualization of software evolution, in: Proceedings of the 12th Annual Conference of the Advanced School for Computing and Imaging. Lommel, Belgium, June 14–16, 2006, pp. 114–121.

[S127] S.G. Eick, J.L. Steffen, E.E. Sumner Jr, Seesoft – A tool for visualizing line oriented software statistics, IEEE Transactions on Software Engineering, 18(11) (1992) 957–968.

[S128] Mockus, S.G. Eick, T.L. Graves, A.F. Karr, On measurement and analysis of software changes, IEEE Transactions on Software Engineering, 1999.

[S129] R. Robbes, M. Lanza, A change-based approach to software evolution. Electron, Electronic Notes in Theoretical Computer Science, 166 (2007) 93–109.

[S130] M. Pinzger, M. Fischer, H. Gall, Towards an integrated view on architecture and its evolution, in: Electronic Notes in Theoretical Computer Science, 2005.

[S131] M. Pinzger, M. Fischer, H. Gall, Towards an integrated view on architecture and its evolution, in: SETra'04: Proceedings of the Software Evolution through Transformations: Model-based vs. Implementation-level Solutions, Elsevier Science Publishers, p. 2005, 2004.

[S132] L. Voinea, A. Telea, CVSGrab: Mining the history of large software projects, in: IEEE VGTC Symposium on Visualization, 2006, pp. 187–194.

[S133] C.F.J. Lange, M.A.M. Wijns, M.R.V. Chaudron, Towards task-oriented modeling using UML, in: Proceedings of the 10th QAOOSE Workshop, co-located with ECOOP'06, 2006.

[S134] H.S.G. Langelier, P. Poulin, Animation coherence in representing software Evolution, in: Proceedings of the 10th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 2006,

[S135] M. Fischer, H. Gall, MDS-Views: Visualizing problem report data of large scale software using multidimensional scaling, in: ELISA'03: International Workshop on evolution of Large-scale Industrial Software Applications, Amsterdam, 2003.

[S136] D. Beyer, Co-change Visualization, in: ICSM'05: Proceedings of the 21st IEEE International Conference on Software Maintenance, Budapest, September 25–30, 2005, pp. 89–92.

[S137] R. Robbes, M. Lanza, M. Lungu, An approach to software evolution based on semantic change, in: FASE'07: Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering, Berlin, Heidelberg, 2007, pp. 27–41.

[S138] O. Greevy, S. Ducasse, T. Gîrba, Analyzing software evolution through feature views: Research articles, Journal of Software Maintenance and Evolution, 18 (6) (2006) 425–456.

[S139] T. Rotschke, R Krikhaar, Architecture analysis tools to support evolution of large industrial systems, in: ICSM'02: Proceedings of the International Conference on Software Maintenance, IEEE Computer Society, Washington, DC, USA, 2002, p. 182.

[S140] M. D'Ambros, M. Lanza, Applying the evolution radar to PostgreSQL, in: MSR'06: Proceedings of the 2006 International Workshop on Mining Software Repositories, ACM, New York, NY, USA, 2006, pp. 177–178.

[S141] B. Kerr, L. Cheng, T. Sweeney, Growing Bloom: Design of a visualization of project evolution, in: CHI'06: Extended Abstracts on Human Factors in Computing Systems, ACM, New York, NY, USA, 2006, pp. 93–98.
[S142] D. Beyer, Co-change visualization applied to PostgreSQL and ArgoUML, in: MSR'06: Proceedings of the 2006 International Workshop on Mining Software Repositories, ACM, New York, NY, USA, 2006, pp. 165–166.
[S143] S. Jucknath-John, D. Graf, Icon graphs: Visualizing the evolution of large class models, in: SoftVis'06: Proceedings of the 2006 ACM symposium on Software Visualization, ACM, New York, NY, USA, 2006, pp. 167–168.
[S144] L. Voinea, A. Telea, M.R.V. Chaudron, Version-centric visualization of code evolution, in: EuroVis'05: IEEE VGTC Symposium on Visualization, Eurographics Association, 2005, pp. 223–230.
[S145] J.P.D. Keast, M.G. Adams, M.W. Godfrey, Visualizing architectural evolution, in: SCE'99: Proceedings of Workshop on Software Change and Evolution, 1999.
[S146] R. Robbes, M. Lanza, Change-based software evolution, in: ERCIM Workshop on Challenges in Software Evolution, 2006, pp. 156–164.

References

[1] S. Alam, S. Boccuzzo, R. Wettel, P. Dugerdil, H. Gall, M. Lanza, EvoSpaces - multi-dimensional navigation spaces for software evolution, in: Human Machine Interaction, Springer- Verlag, Berlin, Heidelberg, 2009, pp. 167–192.
[2] V.R. Basili, H.D. Rombach, The TAME project: towards improvement-oriented software environments, IEEE Transactions on Software Engineering 14 (6) (1988) 758–773. USA.
[3] V.R. Basili, M. Lindvall, M. Regardie, C. Seaman, J. Heidrich, J. Münch, D. Rombach, A. Trendowicz, Linking software development and business strategy through measurement, Computer 43 (4) (2010) 57–65.
[4] V.R. Basili, D. Weiss, A methodology for collecting valid software engineering data, IEEE Transactions on Software Engineering 10 (3) (1984) 728–738.
[5] J. Buckley, Requirements-based visualization tools for software maintenance and evolution, Computer 42 (4) (2009) 106–108.
[6] D. Budgen, M. Turner, P. Brereton, B. Kitchenham, Using mapping studies in software engineering, in: PPIG'08: 20th Annual Meeting of the Psychology of Programming Interest Group, Lancaster University, 2008, pp. 195–204.
[7] G.F. Carneiro, M. Silva, L. Mara, E. Figueiredo, C. Sant'anna, A. Garcia, M. Mendonça, Identifying code smells with multiple concern views, in: SBES '10 Proceedings of the 2010 Brazilian Symposium on Software Engineering, Salvador, Brazil, 2010, pp. 128–137.
[8] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 476–493.
[9] S. Diehl, Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software, Springer-Verlag, New York, 2007.
[10] D. Draheim, L. Pekacki, Process-centric analytical processing of version control data, in: IWPSE'03: Proceedings of the 6th International Workshop on Principles of Software Evolution, IEEE Computer Society, Washington, DC, USA, 2003, pp. 131–140.
[11] S.G. Eick, T.L. Graves, A.F. Karr, J.S. Marron, A. Mockus, Does code decay? assessing the evidence from change management data, IEEE Transactions on Software Engineering 27 (1) (2001) 1–12.
[12] N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, second ed., PWS Publishing Co., Boston, MA, USA, 1998.
[13] H. Gall, K. Hajek, M. Jazayeri, Detection of logical coupling based on product release history, in: ICSM'98: Proceedings of the International Conference on Software, Maintenance, 1998, pp. 190–198.
[14] T. Graves, A. Karr, J. Marron, H. Siy, Predicting fault incidence using software change history, IEEE Transactions on Software Engineering 26 (2000) 653–661.
[15] J. Heer, B. Shneiderman, Interactive dynamics for visual analysis, Communication ACM 55 (4) (2012) 45–54.
[16] I. Herman, G. Melançon, M.S. Marshall, Graph visualization and navigation in information visualization: a survey, IEEE Transactions on Visualization and Computer Graphics 6 (1) (2000) 24–43.
[17] M. Hitz, B. Montazeri, Chidamber and Kemerer's metrics suite: a measurement theory perspective, IEEE TSE 22 (4) (1996) 267–271.
[18] C. Izurieta, J. Bieman, A Multiple Case Study of design pattern decay, grime, nd rot in evolving software systems, Software Quality Journal (2012).
[19] J.H. Jahnke, H.A. Muller, A. Walenstein, N. Mansurov, K. Wong, Fused data-centric visualizations for software evolution environments, in: IWPC'02: Proceedings of the 10th International Workshop on Program Comprehension, IEEE Computer Society, Washington, DC, USA, 2002, pp. 187–196.
[20] B. Johnson, B. Shneiderman, Tree-Maps: a space-filling approach to the visualization of hierarchical information structures, in: Gregory M. Nielson, Larry Rosenblum (Eds.), VIS'91: Proceedings of the 2nd Conference on Visualization, IEEE Computer Society Press, Los Alamitos, CA, USA, 1991, pp. 284–291.
[21] D.A. Keim, H.-P. Kriegel, Visualization techniques for mining large databases: a comparison, IEEE Transactions on Knowledge and Data Engineering 8 (6) (1996) 923–938.
[22] H. Kagdi, M.L. Collard, J.I. Maletic, A survey and taxonomy of approaches for mining software repositories in the context of software evolution, Journal of Software Maintenance and Evolution 19 (2) (2007) 77–131.
[23] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007.
[24] G. Langelier, H. Sahraoui, P. Poulin, Visualization-based analysis of quality for large-scale software systems, in: ASE '05: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, ACM, New York, NY, USA, 2005, pp. 214–223.
[25] M. Lanza, S. Ducasse, Polymetric views – a lightweight visual approach to reverse engineering, IEEE Transactions on Software Engineering 29 (9) (2003) 782–795.
[26] M. Lanza, R. Marinescu, Object-Oriented Metrics in Practice, Springer-Verlag, New York, 2005.
[27] A.M.J.I. Maletic, M.L. Collard, A task oriented view of software visualization, in: VISSOFT '02: 1ST International Workshop on Visualizing Software for Understanding and Analysis, IEEE CS Press, 2002, pp. 32–40.
[28] R. Mazza, Introduction to Information Visualization, Springer-Verlag, London, 2009.
[29] R.L. Novais, C. Nunes, C. Lima, E. Cirilo, F. Dantas, A. Garcia, M. Mendonça, On the proactive and interactive visualization for feature evolution comprehension: an industrial investigation, in: Proceedings of 34th International Conference on Software Engineering, Zurich, 2012, pp. 1044–1053.
[30] R.L. Novais, M.G. Mendonça Neto, D.L. Maron, I.C. Machado, C.A.N. Lima, On the use of a multiple-visualization approach to manage software bugs, in: WBVS'11: I Brazilian Workshop on Software Visualization, São Paulo, 2011.
[31] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: EASE'08: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, University of Bari, Italy, 2008.
[32] B. Shneiderman, The eyes have it: A task by data type taxonomy for information visualizations, in: VL'96: Proceedings of IEEE Symposium on Visual Languages, IEEE Computer Society, Washington, DC, USA, 1996, p. 336.
[33] R. Tamassia, G.D. Battista, C. Batini, Automatic graph drawing and readability of diagrams, IEEE Transactions on Systems, Man, and Cybernetics 18 (1) (1988) 61–79.
[34] A. Telea, A. Maccari, C. Riva, An open visualization toolkit for reverse architecting, in: IWPC'02: Proceedings of the 10th International Workshop on Program Comprehension, IEEE Computer Society, Washington, DC, USA, 2002, pp. 3–11.
[35] C. Ware, Information Visualization: Perception for Design, second ed., Morgan Kaufmann, San Francisco, CA, 2004.
[36] R. Wettel, M. Lanza, R. Robbes, Software systems as cities: a controlled experiment, in: ICSE'11: Proceedings of 33rd International Conference on Software Engineering, IEEE CS Press, 2011, pp. 551–560.
[37] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.
[38] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
[39] H. Hata, O. Mizuno, T. Kikuno, Bug prediction based on fine-grained module histories, in: 34th International Conference on Software Engineering (ICSE), 2012, vol., no., pp. 200,210, 2–9 June 2012.
[40] E. Giger, M. D'Ambros, M. Pinzger, H.C. Gall, Method-level bug prediction, in: ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2012, vol., no., pp. 171,180, 20–21 Sept. 2012.
[41] M. Ribeiro, H. Pacheco, L. Teixeira, P. Borba, Emergent feature modularization, in: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (SPLASH '10), ACM, New York, NY, USA, pp. 11–18.
[42] A. Olszak, B.N. Jørgensen, Modularization of legacy features by relocation and reconceptualization: how much is enough? in: 16th European Conference on Software Maintenance and Reengineering (CSMR), 2012, vol., no., pp. 171,180, 27–30 March 2012.
[43] B. Merkle, Stop the software architecture erosion: building better software systems, in: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (SPLASH '10)., ACM, New York, NY, USA, 2010, pp. 129–138.
[44] S.-L. Voinea, Software Evolution Visualization, Ph.d. Thesis, Technische Universiteit Eindhoven, Eindhoven, NL, 2007.
[45] G.de.F. Carneiro, R. Magnavita, M. Mendonça, Combining software visualization paradigms to support software comprehension activities, in: Proceedings of the 4th ACM Symposium on Software Visualization (SoftVis '08)., ACM, New York, NY, USA, 2008, pp. 201–202.
[46] G.de F. Carneiro, M. Mendonça, R. Magnavita, An experimental platform to characterize software comprehension activities supported by visualization, in: 31st International Conference on Software Engineering – Companion Volume, 2009. ICSE-Companion 2009, vol., no., pp. 441,442, 16–24 May 2009.

[47] G.de F. Carneiro, M. Mendonça, R. Magnavita, Proposing a visual approach to support the characterization of software comprehension activities, in: IEEE 17th International Conference on Program Comprehension, 2009. ICPC '09., vol., no., pp. 291,292, 17–19 May 2009.

[48] R. Novais, G.F. Carneiro, P. Simões Júnior, M. Medonça, On the use of software visualization to analyze software evolution: an interactive differential approach, in: R. Zhang, J. Zhang, Z. Zhang, J. Filipe, J. Cordeiro (Eds.), Enterprise Information Systems, Lecture Notes in Business Information Processing, vol. 102, Springer, Berlin Heidelberg, 2012, pp. 241–255. doi:10.1007/978-3-642-29958-216..

[49] R. Novais, P. Simões Júnior, M. Mendonça, Timeline matrix: an on demand view for software evolution analysis, in: Software Visualization (WBVS), 2012 2nd Brazilian Workshop on, 2012, pp. 1–8.