Software Visualizations to Analyze Memory Consumption: A Literature Review

ALISON FERNANDEZ BLANCO, ISCLab, Department of Computer Science (DCC), University of Chile, Chile
ALEXANDRE BERGEL, ISCLab, Department of Computer Science (DCC), University of Chile, Chile
JUAN PABLO SANDOVAL ALCOCER, Department of Computer Science, School of Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile

Understanding and optimizing memory usage of software applications is a difficult task, usually involving the analysis of large amounts of memory-related complex data. Over the years, numerous software visualizations have been proposed to help developers analyze the memory usage information of their programs.

This paper reports a systematic literature review of published works centered on software visualizations for analyzing the memory consumption of programs. We have systematically selected 46 articles and categorized them based on the tasks supported, data collected, visualization techniques, evaluations conducted, and prototype availability. As a result, we introduce a taxonomy based on these five dimensions to identify the main challenges of visualizing memory consumption and opportunities for improvement. Despite the effort to evaluate visualizations, we also find that most articles lack evidence regarding how these visualizations perform in practice. We also highlight that few articles are available for developers willing to adopt a visualization for memory consumption analysis. Additionally, we describe a number of research areas that are worth exploring.

CCS Concepts: • **Human-centered computing** → *Empirical studies in visualization*; **Visualization systems and tools**; **Visualization techniques**; **Empirical studies in visualization**; • **General and reference** → **Surveys and overviews**; Performance.

Additional Key Words and Phrases: systematic literature review, software visualization, memory consumption

ACM Reference Format:
Alison Fernandez Blanco, Alexandre Bergel, and Juan Pablo Sandoval Alcocer. 2021. Software Visualizations to Analyze Memory Consumption: A Literature Review. 1, 1 (September 2021), 33 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Software development often involves deep and intricate technical aspects. Execution time and memory consumption are two primary resources to consider in software engineering [86, 88]. Keeping the amount of memory consumed by a software system under control is an example of such a programming challenge.

Understanding software execution. Manually understanding and addressing memory issues is challenging since it usually involves analyzing several metrics at once and requires a thorough analysis of the respective code [9, 13]. To assist developers in this activity, software development environments provide tools to monitor and report resource usage during software execution. An example of such tools is the execution profiler, designed to report information

Authors' addresses: Alison Fernandez Blanco, ISCLab, Department of Computer Science (DCC), University of Chile, Chile; Alexandre Bergel, ISCLab, Department of Computer Science (DCC), University of Chile, Chile; Juan Pablo Sandoval Alcocer, Department of Computer Science, School of Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile.

Manuscript submitted to ACM

about the behavior exhibited of a target program during its execution. Profilers help developers evaluate how well programs perform based on a set of dynamic aspects, including memory consumption, garbage collections, execution time, and frequency of function calls [3, 7, 47]. These metrics are usually displayed through full-text reports or textual tables, affecting the process and leading to actionable conclusions.

Visualizing memory consumption. Over the years, the research community of software visualization has proposed a variety of visualizations to support software comprehension [25, 43]. It has also been shown that interactive visualization reinforces the cognition that facilitates human interaction to explore and understand data [94]. Due to this, software visualizations enriched with interaction mechanisms become a powerful alternative for displaying profiler reports to support developers in understanding and addressing memory-related issues. Each one of these visualizations provides a wide spectrum of metrics (*e.g.,* number of created objects [34, 62], memory access [31]), and data representations (*e.g.,* call graphs [34, 98] and call context trees [9]).

This paper presents a systematic literature review of software visualizations to analyze memory consumption. We initially used keyword searches against three popular scientific databases and complemented it with a bi-directional snowballing and a manual search of relevant venues. As a result, we found 420 articles published without counting duplicates. From these, we selected 46 articles based on inclusion/exclusion criteria and quality assessment. In this way, we included only the studies centered on visualizations to analyze the memory consumption of a software program. As a consequence, we excluded articles that only focus on memory issues without visualization, articles that analyze the memory used by the visualization per se, and articles that focus on other performance metrics excluding memory usage. In summary, our systematic review focuses on published works centered on visualizations that assist practitioners in examining memory usage to identify optimizations opportunities.

Section 2 presents the methodology we followed in this study. Section 3 displays the main findings by answering the questions defined in Section 2.1. Section 4 provides the open challenges for the new visualizations centered on analyzing memory usage. Section 6 discusses the state-of-art. Section 5 discusses the threats to validity of this study, and Section 7 exposes the future work and the conclusions.

2 METHODOLOGY

Overview. This literature review follows a systematic and rigorous methodology to identify and categorize literature related to memory consumption visualization. We use a seven high-level steps methodology inspired by well-recognized software engineering guidelines for systematic reviews [41, 42]. Our steps are:

(1) Define Research Questions
(2) Develop a Search strategy
(3) Define Inclusion and Exclusion Criteria
(4) Screen and Select Studies for Inclusion
(5) Quality Assessment
(6) Data Extraction
(7) Analysis

We describe each one of these steps in the following sections.

2.1 Research Questions

The purpose of this literature review is to inspect, analyze, and discuss the state-of-art regarding software visualizations focused on helping developers to understand memory consumption. In particular, we are interested in addressing the research questions described in Table 1. We believe that answering these research questions (RQ) will assist future researchers in creating new visualizations focused on supporting developers during memory consumption analysis.

Table 1. Research Questions

Research Question	Dimension & Rationale
RQ1: *Which tasks are supported by the software visualizations to help users with the analysis of memory consumption?*	*Problem Domain:* Identify the tasks that software visualization targets to facilitate during the memory consumption analysis. For instance, identify bottlenecks or detect memory leaks.
RQ2: *What aspects of the software are abstracted by the software visualizations to help users with the analysis of memory consumption?*	*Data:* Software visualizations display large amounts of data (*e.g.*, memory allocations, memory accesses) extracted from the execution or code of software applications. This information allows developers to understand the memory consumption of a program.
RQ3: *Which software visualizations have been proposed to help users with the analysis of memory consumption?*	*Visual Representation:* The use of different visual techniques to abstract complex and related data is an important topic. The way on which visual elements are rendered and presented to the user is also relevant because it may impact how the user interacts and perceives the visualization. In particular, we are interested in reviewing: **RQ3.1:** *Which visual techniques are used?*, **RQ3.2:** *Which interaction tasks are supported?*, and **RQ3.3:** *Where are the visual elements rendered?*
RQ4: *How are software visualizations to help users with the analysis of memory consumption evaluated?*	*Evaluation:* Analyzing how software visualization is evaluated provides (i) an overview of the proposed visualization's effectiveness and usefulness and (ii) a better understanding of conducted evaluation strategies.
RQ5: *What software visualization tools or prototypes are available to help users with the analysis of memory consumption?*	*Availability:* The availability of a prototype or tool is an opportunity (i) for practitioners to benefit from the approach and (ii) for researchers to replicate the results or complementing the associated research articles.

Dimensions. Our research questions focus on five dimensions: problem domain, data, visual representation, evaluation, and artifact. The research questions and their dimensions were inspired by six surveys [46, 49, 51, 64, 65, 74]. These studies present a number of relevant dimensions to give an enriched overview of software visualizations. Table 2 shows the six surveys mentioned previously with their respective dimensions and how they are related to our research questions.

RQ1 centers on software engineering tasks supported by the visualization. **RQ1** was inspired by three previous studies: Price *et al.* [64, 65] provide taxonomies with a minor summary about the intention of the visualizations on their *Purpose* dimension. Maletic *et al.* [49], and Merino *et al.* [51] consider general tasks of software engineering like reverse

Table 2. An overview to the relations between our dimensions and the dimensions proposed by some works of the state-of-art.

Survey	RQ1	RQ2	RQ3	RQ4	RQ5
Price *et al.* [64, 65]	Purpose	Scope and content	Form, method, interaction and effectiveness	Empirical evaluation	-
Roman *et al.* [74]	-	Scope and abstraction	Specification, method, interface and presentation	-	-
Maletic *et al.* [46]	Task	Target	Representation and medium	-	-
Merino *et al.* [51]	Task	Data source	Representation and medium	-	Tool

engineering, maintenance, and testing. Compared to these works, our *Problem Domain* dimension focuses on detailed software engineering tasks related to memory usage.

In the case of **RQ2** and **RQ3**, the surveys mentioned previously present detailed information for these dimensions, providing an analysis of the collected data and how this data is abstracted visually to the user. In this literature review, our *Data* dimension describes the metrics considered for the analysis of memory usage, and the *Visual Representation* dimension reports the visual encodings, interactions, and medium used by the approaches.

Our study also includes two dimensions: **RQ4** and **RQ5** corresponding to evaluation and availability. **RQ4** was only covered by Price *et al.* [64, 65], and **RQ5** by Merino *et al.* [51]. We include both dimensions since they are relevant in the research community to understand how the visualizations were evaluated and if they may be replicable.

2.2 Search Strategy

Initial manual search. According to the Systematic Literature Review guidelines [42, 106], before performing an automatic search phrase and defining an inclusion/exclusion criteria, it is necessary to search for an initial set of relevant articles. To do this, we manually reviewed the articles published between 2017 and 2020 in the following scientific venues:

- *IEEE International Working Conference on Software Visualization (VISSOFT)*
- *International Symposium on Memory Management (ISMM)*

We selected these conferences because the articles dedicated to software visualization and memory management present a sound corpus for our study. Besides, these conferences are classified respectively in the good (B) and excellent (A) category according to CORE rankings[1], which determines conference rankings based on a mix of indicators (*e.g.,* citation rates, paper submission, acceptance rates). The result of our initial manual search ends up with five articles [9, 11, 31, 77, 102]. We used these papers as a base to define our search strategy by extracting search terms derived from the research questions.

Search phrase development. We extracted the search terms that fit our scope of the title, abstract, and keywords from the articles found at the initial manual search. Furthermore, we expanded these search terms with synonyms and alternatives as shown on Table 3.

To find potential articles considered for our study, we combined these terms into a query as it follows:

[1] https://www.core.edu.au/conference-portal

Table 3. Search terms and alternatives of spelling

Term	Alternatives
Memory*	memory heap, memory allocation, memory consume, memory consumption, memory usage, memory management, memory issues, memory issue, memory bloats, memory leaks, memory access, memory address
Visual*	visualize, visualization, visualisation, visual, visuals, visualizations, visualisations
Software*	software, program, application

Memory AND Visual* AND Software**

The previous query represents the condition that an article should meet to be considered in our study. We executed the query against the *abstract*. We did not limit the search based on publication date to find the most significant number of relevant articles for our study. We performed the search over three digital libraries:

- *ACM Digital Library*
- *IEEE Xplore*
- *Scopus*

As a result, we found 533 papers that meet these criteria, including our initial set of five papers. The latter gives a level of certainty that we could find any article that proposes a visualization to assist developers with memory usage analysis. However, we may have a number of false positives that we detected in the following steps. Additionally, appendix A presents the search strings used for each digital library mentioned before.

Additional manual paper selection. In the previous phase, we found articles that contain the keywords used in the query search. As a result, we located articles that may be useful and representative. However, we may have missed some relevant articles. For instance, articles that use more particular memory-related keywords (*e.g.*, cache, fragmentation) may or may not be considered by our query. Therefore, in order to not miss any related paper, we also performed a manual search on the last ten editions (2010-2020) from the following venues:

- *IEEE International Working Conference on Software Visualization (VISSOFT), the continue of IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT) and ACM Symposia on Software Visualization (SOFTVIS)*
- *International Symposium on Memory Management (ISMM)*

We selected these conferences because we noticed that most of the articles resulting from our automatic phrase search were published in them. We also reviewed only articles published in the last ten issues due to our time and human resources. In total, around 305 articles were published in these venues over the last ten editions. We manually reviewed each article based on its title and abstract. Consequently, we found ten articles that fall within the scope of this literature review. However, seven articles were found in the earlier phases. Therefore, we identified three additional articles during this phase.

Bi-directional snowballing. We performed a backward and forward snowballing over the ten articles found in the previous phase to complete our search. The snowballing procedure consists of identifying additional studies using the system of references between articles [105]. For this reason, we checked the references in each article, and we reviewed

the list of articles that reference any article of our selection. Thus, we could add relevant research published after or before the publication date of our selection set by performing several iterations until non-relevant papers are referenced. We then selected *Google Scholar* to perform the forward snowballing due to the facilities provided to select the papers that cite a specific one. On the other hand, the backward snowballing was performed manually. Consequently, over two iterations, we found 56 additional articles that could be considered in this study, collecting a total of 420 papers without counting duplicated articles.

2.3 Inclusion & Exclusion Criteria

We elaborated inclusion and exclusion criteria based on the scope of this study. Table 4 details the inclusion and exclusion criteria. In particular, we are interested in papers that use visualization techniques to help developers understand and address memory issues.

Table 4. Inclusion and Exclusion Criteria

Inclusion Criteria
• *I1:* Papers published in a peer reviewed journal, conference or workshop on data visualization, computer science, or computer engineering. • *I2:* Papers written in English.
Exclusion Criteria
• *E1:* Papers that focus on other performance metrics (*e.g.,* execution time). • *E2:* Papers that only study memory issues *or* visualization issues. • *E3:* Posters, keynotes, challenges and previous papers that only introduce the idea of most recent full papers (*e.g.,* short papers).

The three authors of this study performed a revision of 420 articles based on inclusion/exclusion criteria. The three authors independently read and analyzed the title, abstract, keywords, and venue to decide if an article is excluded or not. However, if an author did not have enough information to decide, the author should read the introduction and conclusions of the article. Next, each author responds independently if an article should be included or not using a spreadsheet that lists the 420 articles.

Then, we examined the spreadsheet responses to calculate the kappa of Fleiss for the inter-rater reliability [28]. As a result, we got 0.72 for the Kappa Fleiss analysis, which is generally considered a good agreement beyond chance [29]. We also identified 38 articles on which we have discrepancies in the spreadsheet responses. Most of these differences were related to E1 and E2 criteria. For instance, some articles focus on using a software visualization to understand the trace execution of programs, but not explicitly center on memory consumption. On the other hand, other articles are dedicated to analyzing memory problems, but not primarily with software visualizations.

To resolve all conflicts, the three authors conducted a second review of the 38 articles, analyzing the full content of each article. The authors then had a discussion session to develop an agreement based on the responses from the second review. As a result, a total of 49 articles are candidates to be included in our study.

2.4 Quality Assessment

This phase involves the selection of the papers based on their quality [42, 63]. To exclude the articles with insufficient information to contribute to this study, we examine the theoretical contribution, and the experimental evaluation with the checklist used in software engineering surveys [61, 95] detailed in Table 5.

Table 5. Quality assessment adopted from [95]

#	Questions
	Theorical contribution
1	Is at least one of the research questions addressed?
2	Was the study designed to address some of the research questions?
3	Is a problem description for the research explicitly provided?
4	Is the problem description for the research supported by references to other work?
5	Are the contributions of the research clearly described?
6	Are the assumptions, if any, clearly stated?
7	Is there sufficient evidence to support the claims of the research?
	Experimental evaluation
8	Is the research design, or the way the research was organized, clearly described?
9	Is a prototype, simulation or empirical study presented?
10	Is the experimental setup clearly described?
11	Are results from multiple different experiments included?
12	Are results from multiple runs of each experiment included?
13	Are the experimental results compared with other approaches?
14	Are negative results, if any, presented?
15	Is the statistical significance of the results assessed?
16	Are the limitations or threats to validity clearly stated?
17	Are the links between data, interpretation and conclusions clear?

In this step, the three authors assess the quality of each paper based on the checklist mentioned before. Each author assigns a score to every question in the checklist. The score has a numeric scale of three levels: yes (2 points), partial (1 point), and no (0 points). The final score of a paper is measured by summing up the score of all questions. Since the form has 17 questions, the total score of the articles varies from 0 to 34.

Additionally, we follow the criteria of Usman *et al.* [95] by using the lower quartile ($34/4 = 8.5$) as the limit point for including an article based on quality. As a result, all the articles with a score above 8.5 points were considered relevant hence they present enough information to address our research questions.

In total, 35 articles met the quality assessment with the approval of three authors, while 14 articles were detected as discrepancies. Consequently, a second pass was made over these 14 articles. At the second pass, each reviewer independently read and examined the quality assessment of each article again. We then moved on to a discussion session between the three reviewers to resolve conflicts by consensus. Finally, with the second pass, a total of 46 articles were selected to be included in the literature review.

2.5 Data Extraction

To extract the necessary data, the first author of this survey was in charge of examining each of the 46 articles. From each article, she collected general information (*e.g.,* title, publication year, venue) and information according to the

dimensions and rationale of the research questions. Although the data extractor reviewed the entire document, she focused on many particular sections in order to answer the research questions:

- *RQ1 – Problem Domain:* Abstract, introduction, evaluation, conclusion.
- *RQ2 – Data:* Data collection, data extraction, profiling information.
- *RQ3 – Visual Representation:* Visualization, detailed view, visual design, display.
- *RQ4 – Evaluation:* Evaluation, case study, applications, usage scenario.
- *RQ5 – Availability:* Visualization, implementation, conclusion.

The data extractor was also careful to search for data to respond *RQ5* because sometimes artifacts or data sets are placed as a reference or footnotes.

In order to validate the data extraction, the other two authors of this study checked the data to confirm that extraction was correct. The three authors discussed and resolved any disagreements by reviewing the articles and data extraction forms. We then recorded the final data value for data analysis.

We noticed that some articles do not present information to respond to all the research questions during this phase. For example, some articles lack information about the interactions supported, the medium used, or the evaluation conducted. We discussed the data synthesis of these cases in Section 2.6 and Section 3.

2.6 Data Analysis

This section describes the data analysis methods conducted to answer our research questions.

Thematic analysis. We opted to conduct a thematic analysis [90] for RQ1 and RQ2 since we noted that the proposed classification schemes from previous software visualization surveys were general for helping us answer these research questions. In order to create a classification scheme, the first author conducted the thematic analysis following a number of specific steps:

- *Familiarization.* Extracted data is read and reread to have an overview of the information.
- *Generating codes.* The author in charge of the analysis assigned codes that reflect relevant features to answer the research questions. For example, the author assigned the code *"Detection of memory fragmentation"* for the text: "To help the user find potential memory fragmentation problems, we display memory blocks that have been freed exactly one time and not reused" [73]. Additionally, continuous reviews were conducted to refine codes and determine if they were assigned correctly. The latter requires comparing two text segments assigned to the same code to inspect if they reflect the same feature.
- *Constructing initial themes.* All codes are compiled with their associated data into coherent groups to identify initial themes (broader patterns) that help address the respective research questions. The codes that seem to not belong to a specific theme were grouped as miscellaneous and analyzed in the next step.
- *Reviewing themes.* Initial themes were checked against the associated data (*e.g.,* segments of text) and refined to create a final set of themes.
- *Defining and naming themes.* Each theme of the final set was defined with a detailed description and an informative name. For instance, the themes for RQ2 are generated based on the source of the data abstracted (*e.g.,* data from program execution, data from source code, data from versions).

Finally, the remaining two authors checked the process by reviewing the consistency of codes and themes against the associated data and examining if the themes created respond to RQ1 and RQ2. Three meetings were held involving

the three authors to discuss the disagreements or potential issues of the generated codes and themes. As a consequence, we minimized potential inconsistencies in the coding process.

Content analysis. We conducted a deductive content analysis [22, 27] to answer RQ3.1, RQ3.2, and RQ4 since the data synthesis was performed based on defined classification schemes from previous studies shown in Table 6.

Table 6. Classification scheme

ID	Dimension	Classification scheme	Proposed by
RQ3.1	Visual techniques	Geometrically-transformed displays, iconic displays, dense pixel displays, stacked displays and standard 2D/3D displays	Keim [38]
RQ3.2	Interactions	Select, explore, reconfigure, encode, abstract/elaborate, filter and connect	Yi and colleagues [107]
RQ4	Evaluation	No explicit evaluation, empirical, theoretical	Merino and colleagues [50]

For RQ3.2, we classified only the articles that present information to answer the research question. We exposed the number of articles that lack data to answer these research questions. For RQ4, Merino and colleagues proposed a category of "No explicit evaluation" for these cases.

The classification of articles for RQ3.1, RQ3.2, and RQ4 was performed by two authors independently. Each author filled a spreadsheet to classify the articles based on a detailed description of the predetermined categories. Later, to check the agreement between reviewers, we calculated metrics for reliability (Cohen kappa [99], percentage of agreement). As a result, we noticed that reviewers present a "substantial agreement" (kappa > 0.61) and a percentage agreement above 80% at classifying the articles for most categories. However, we noted disagreements on the classification based on the interactions supported and the evaluations conducted (case study vs. usage scenario). We discussed the disagreements in meetings by exposing the data and examining the description of the categories. Consequently, we resolved the discrepancies and advanced to expose the results to answer the research questions.

Finally, to respond to RQ3.3 we only reviewed the medium employed. And for RQ5, we listed the link for the prototype and the additional information (video, sample data) provided in the link.

3 RESULTS

3.1 Overview

Table 7 summarizes the results of all steps in our systematic search methodology. It shows different stages of the process for search and selected relevant articles. *Unique* columns show the number of non-duplicated articles. As a result, we found that 32.27% of the articles found in the search phase over digital libraries were duplicated articles.

We also noticed that 11.66% of the articles found during the search phases were selected based on the inclusion/exclusion criteria.

In the end, Table 8 and Table 9 display the 46 articles that passed our inclusion/exclusion criteria and satisfied the criteria of our quality assessment. A set of collected data from the 46 articles is available online[2].

[2] https://www.dropbox.com/s/7srvxiacftg2pm1/ArticleClassification.csv?dl=0

Table 7. Systematic Search Results

Source	Date	Search Results	Unique	Inclusion/Exclusion Criteria	Quality Assessment	Included
ACM DL	March 18, 2021	209				
IEEE	March 18, 2021	72				
SCOPUS	March 18, 2021	252				
Search phrase		533	361	27	24	24
Additional manual search		3	3	3	3	3
Bi-directional snowballing		56	56	19	19	19
					Total	46

Publication year. During the search phase in digital libraries, we do not limit the search based on publication dates. We also do not exclude articles based on their publication date during the selection phase. However, we noted that the articles considered by our study were published between 1996 and 2020. Furthermore, we detected that the number of published articles increases over time with high picks (4 articles published) in 2002, 2010, and 2018.

Venues. Regarding the distribution of articles based on the venue, we identified 27 different venues where the papers were published. Most of the venues are related to software visualizations, software maintenance and software comprehension. Furthermore, we observed that 32.60% of selected studies were published in software visualization conferences (VISSOFT and SOFTVIS). We also noticed that 17.39% of the articles were published in journals usually involved with computer graphics and visualizations (*e.g.,* Computer Graphics Forum, IEEE TVCG). Finally, the remaining articles were published in various conferences and workshops, usually related to software maintenance and software comprehension (*e.g.,* ISMM, ICSME).

3.2 RQ1: Problems domain

Selected articles propose software visualizations that usually target to help developers perform debugging and performance tasks. We performed thematic analysis to find patterns over the data to provide details of which tasks are supported by these visualizations. As a result, we detected themes that help users adopt a suitable software visualization according to their requirements. We classified the visualizations based on (i) focus point analysis and (ii) issue detection. Table 10 shows the distribution of papers based on this classification. According to our classification, a visualization could focus on analyzing a specific point and detecting multiple memory issues. As a result, a visualization could belong to multiple categories.

Focus point analysis. Articles explain why the proposed visualization is helpful in different sections. During our thematic analysis, we noticed that a number of articles present a general description by specifying that the proposed visualization has a general purpose in helping developers understand and monitor an application's memory consumption. On the other hand, we found articles that propose dedicated visualizations that allow users to analyze specific points (*e.g.,* data structure, cache behavior). We classified the articles based on the focus point analysis described below.

Table 8. The included papers in the study (S1-S29)

ID	Title	Venue	Year	Ref.
S1	Evaluating an Interactive Memory Analysis Tool: Findings from a Cognitive Walkthrough and a User Study	*PACMHCI*	2020	[101]
S2	Memory Cities: Visualizing Heap Memory Evolution Using the Software City Metaphor	*VISSOFT*	2020	[102]
S3	PVC.js: visualizing C programs on web browsers for novices	*Heliyon*	2020	[35]
S4	Enhancing Commit Graphs with Visual Runtime Clues	*VISSOFT*	2019	[77]
S5	Visual performance analysis of memory behavior in a task-based runtime on hybrid platforms	*CCGRID*	2019	[59]
S6	Detailed heap profiling	*ISMM*	2018	[11]
S7	Effective visualization of object allocation sites	*VISSOFT*	2018	[9]
S8	NumaMMA: NUMA MeMory Analyzer	*ICPP*	2018	[93]
S9	Memaxes: Visualization and analytics for characterizing complex memory performance behaviors	*TVCG*	2018	[30]
S10	Atlantis: Improving the analysis and visualization of large assembly execution traces	*ICSME*	2017	[33]
S11	Visual exploration of memory traces and call stacks	*VISSOFT*	2017	[31]
S12	Leveraging Managed Runtime Systems to Build, Analyze, and Optimize Memory Graphs	*VEE*	2016	[85]
S13	Interactive visualization of cross-layer performance anomalies in dynamic task-parallel applications and systems	*ISPASS*	2016	[24]
S14	Efficiently identifying object production sites	*SANER*	2015	[34]
S15	TABARNAC: Visualizing and resolving memory access issues on NUMA architectures	*VPA*	2015	[6]
S16	Visualization of memory access behavior on hierarchical NUMA architectures	*VPA*	2014	[104]
S17	A visual approach to investigating shared and global memory behavior of CUDA kernels	*Comput Graph Forum*	2013	[75]
S18	Visualizing the allocation and death of objects	*VISSOFT*	2013	[98]
S19	Abstracting runtime heaps for program understanding	*IEEE TSE*	2012	[48]
S20	Topological analysis and visualization of cyclical behavior in memory reference traces	*PacificVis*	2012	[17]
S21	Vasco: A visual approach to explore object churn in framework-intensive applications	*ICSM*	2012	[26]
S22	Abstract visualization of runtime memory behavior	*VISSOFT*	2011	[16]
S23	A map of the heap: Revealing design abstractions in runtime structures	*SOFTVIS*	2010	[58]
S24	Allocray: Memory allocation visualization for unmanaged languages	*SOFTVIS*	2010	[73]
S25	Automated construction of memory diagrams for program comprehension	*ACM SE*	2010	[19]
S26	Heapviz: interactive heap visualization for program understanding and debugging	*SOFTVIS*	2010	[1]
S27	Making Sense of Large Heaps	*ECOOP*	2009	[53]
S28	Visualizing the Java heap to detect memory problems	*VISSOFT*	2009	[71]
S29	Hdpv: Interactive, faithful, in-vivo runtime state visualization for C/C++ and Java	*SOFTVIS*	2008	[87]

Table 9. The included papers in the study (S30-S46)

ID	Title	Venue	Year	Ref.
S30	Interactive Visualization for Memory Reference Traces	*Comput Graph Forum*	2008	[15]
S31	Visualizing dynamic memory allocations	*VISSOFT*	2007	[55]
S32	Visualising dynamic memory allocators	*ISMM*	2006	[12]
S33	Jove: Java as it happens	*SOFTVIS*	2005	[72]
S34	YACO: A User Conducted Visualization Tool for Supporting Cache Optimization	*HPCC*	2005	[69]
S35	Interactive locality optimization on numa architectures	*SOFTVIS*	2003	[56]
S36	Visualizing Java in action	*SOFTVIS*	2003	[70]
S37	GCspy: an adaptable heap visualisation framework	*OOPSLA*	2002	[68]
S38	Visualising the train garbage collector	*ISMM*	2002	[67]
S39	Visualizing memory graphs	*Software Visualization*	2002	[111]
S40	Visualizing the execution of Java programs	*Software Visualization*	2002	[20]
S41	Visualizing the impact of the cache on program execution	*ICIV*	2001	[108]
S42	Visualizing the memory access behavior of shared memory applications on NUMA architectures	*ICCS*	2001	[89]
S43	Visualizing reference patterns for solving memory leaks in Java	*ECOOP*	1999	[21]
S44	A cache visualization tool	*Computer vol. 30*	1997	[96]
S45	DDD — a free graphical front-end for Unix debuggers	*SIGPLAN Not.*	1996	[110]
S46	Monitoring data-structure evolution in distributed message-passing programs	*HICSS*	1996	[78]

Table 10. Classification of articles based on the tasks

Problem domain		References	Total
Focus point analysis	Specific architectures	S5, S8-S9, S12-S13, S15-S17, S35, S42, S46	11
	Data structure	S3, S19, S23, S25-S29, S33, S36, S39-S40, S43, S45	14
	Cache performance	S22, S30, S34, S41, S44	5
	Memory regression	S4	1
	General	S1-S2, S6-S7, S10-S11, S14, S18, S20-S21, S24, S31-S32, S37-S38	15
Issue detection	Memory leak	S1, S2, S24, S26-S29, S40, S43, S46	10
	Memory bloat	S1, S4, S6-S7, S14, S18-S19, S27-S28	9
	Memory churn	S1, S21, S24, S28-S29, S38	6
	Memory fragmentation	S5, S24, S31-S32, S37-S38	6

- *Specific architecture.* We found 23.91% articles dedicated to analyzing the memory consumption of applications with specific architectures (HPC, parallel, embedded, distributed). These articles usually propose visualizations that display how the data is accessed and used by multiple threads and multiple processors. For example, article **S35** [56] allow developers to understand the memory access behavior of parallel NUMA applications. This article proposes a visualization that helps developers identify which nodes perform the memory accesses and detect an opportunity to reduce remote memory accesses.

- *Data structure.* According to Cormen *et al.* [18], a data structure is a way to store, manage and organize data to facilitate access and modifications. There is a variety of data structures employed in software applications (*e.g.,* lists, dictionaries). However, the inefficient usage of data structure and its operations (*e.g.,* adding, removing elements) generates memory issues that affect the performance. For this reason, data structure analysis is a prevalent task during software development. We detected that 30.46% of visualizations support developers in analyzing and inspecting data structures. For instance, article **S26** [1] propose *Heapviz* that allows developers to identify large data structures and which objects are shared by several data structures. *Heapviz* displays a node-link diagram to visualize the references between objects and locate the nodes using a radial layout to use screen space efficiently.
- *Cache performance.* Cache stores data so that future requests for that data can be responded more quickly. Tracking the cache activity in a software application helps developers understand memory performance at a fine-grained level. Accordingly, developers may require and analyze memory access and cache performance; hence, the cache activity's analysis influences detecting memory access anomalies. We found that 10.86% of visualizations support developers in analyzing cache performance. For example, article **S34** [69] propose *YACO* to help users with the analysis of access patterns and cache misses. *YACO* present multiple views to display statistics related to cache performance and allow developers to find data that frequently enter and leave the cache.
- *Memory regression.* Source code changes may impact the performance of an application [3]. Only article **S4** [77] allows developers to analyze the memory variations between code changes. This article proposes *Spark Circle* that enables users to compare two commits based on the number of allocated objects, the execution time, and the number of modified methods. As a consequence, a developer can identify the growth or reduction of allocated objects between commits.
- *General.* As we mentioned before, we found 32.60% of the articles do not focus on a specific point. These articles determine that the goal of the proposed visualization is to analyze memory consumption. Therefore we could not find specific analysis points such as analysis of data structures or cache behavior. During the generation of codes, we detected that these articles usually display memory access or heap usage. Furthermore, most of these articles are useful for detecting memory issues.

Issue detection. We found that 47.82% of the articles propose visualizations that allow developers to identify memory issues. Additionally, we detected several articles that claim to present helpful visualizations to address memory anomalies but do not specify which kind of anomalies or particular situations can be addressed with the proposed visualization. Consequently, we only classified the articles that present a detailed description of the memory issues addressed and how developers could employ the visualization to find these anomalies. We described the memory issues that were found below.

- *Memory leak.* A memory leak is an issue deeply related to improper memory management [10, 54]. A memory leak occurs when an unused memory allocation cannot be released from memory [21]. The latter usually happens because there are allocations that reference an unused allocation. As a consequence, the application can run out of memory and crash. We found that 21.73% of the visualizations help developers in detecting memory leaks. For instance, article **S40** [20] presents a node-link diagram to display the references between the objects not reclaimed by the garbage collector. This visualization allows developers to identify memory leaks by exploring which objects are no longer used but are referenced by other objects.

- *Memory bloat.* Memory bloat exposes inefficient use of memory by a program [101]. A memory overhead significantly affects software applications by reducing their scalability and usability. Developers should notice that an application may be free of memory leaks, but could require excessive memory to operate correctly. According to LaToza *et al.* [44], developers usually ask, *"How big is this in memory?"* and *"How many of these objects get created?"* These questions are related to distinguish memory growth. Addressing memory bloats impacts the application behavior, making it more usable and faster in some cases [37]. We detected that 19.56% of visualizations support developers in identifying excessive memory consumption. For example, article **S1** [101] allows developers to explore and observe memory consumption over time by using multiple views with *AntTracks*. As a result, *AntTracks* assists developers in detecting memory growth and explore suspicious objects allocated over time.
- *Memory churn.* This issue occurs when an application allocates and releases a large number of short-living objects [26]. For example, memory churn can happen if a program allocates several new objects in the middle of nested loops. As a result, the time spent on allocating objects in a heap and the number of garbage collections increases. Thus the application decreases its performance due to frequent garbage collection. We detected that 13.04% of the visualizations help developers in detecting memory churn. For example, article **S21** [26] proposes *Vasco*, a visualization that allows developers to identify where and when an object is allocated and no longer used. The authors of *Vasco* described how they use visualization to reuse some objects and reduce the number of allocations and garbage collections.
- *Memory fragmentation.* This issue related to a failure at reusing memory that has been released. Furthermore, excessive fragmentation over memory may lead to more costly performance behavior. We found that 13.04% of the visualizations help developers identify memory fragmentation. For instance, article **S31** [55] allows users to analyze the behavior of a memory allocator by displaying the memory accesses through time. This visualization enables developers to identify unnecessary fragmentation since free memory blocks can be detected quickly.

We also noticed that selected articles usually define the roles of users of visualizations. We detected that 13.04% of the visualizations help students or novice developers analyze memory consumption. For example, article **S3** [35] presents *PVC* to support students with understanding the program execution status and behavior. The authors of this article experimented with 35 university students to evaluate the usability of *PVC*. Furthermore, most of the visualizations (89.13%) assist developers and software engineers. Some of these articles determine that proposed visualizations are suitable for supporting developers with experience in software engineering and with knowledge of memory management.

3.3 RQ2: Data

Monitoring and analyzing memory usage is a complex task for developers since it is necessary to collect and examine different software aspects. The selected articles usually present detailed sections to describe the data collection. We noticed that several articles implement a profiler to gather information. Other studies use dedicated tools for this purpose, such as *Pin* [45], *Jinsight* [20], *DynamoRIO* [3], *etc.* Nonetheless, some studies only describe the information visualized, but do not explicitly mention how they collect the data.

[3] http://dynamorio.org

We defined the classification scheme according to the sources from which various data were collected. As a result, the articles are categorized based on three sources: (i) program execution, (ii) source code, and (ii) version control systems. In this classification, software visualizations can belong to multiple categories.

Table 11. Classification of articles based on the data source

Data source	Data	References	Total
Program execution	Memory allocations	S4, S7, S14, S19, S21, S23, S25-S27, S33, S39, S45	12
	Memory allocations and release	S1-S2, S12, S18, S28-S29, S31-S32, S36-S38, S40, S43	13
	Memory accesses	S3, S5-S6, S8-S11, S13, S15-S17, S20, S22, S24, S30, S34-S35, S41-S42, S44, S46	21
	Relationships between functions/methods	S1, S6-S7, S10-S11, S14, S21, S40	8
	Variable references	S1-S3, S12, S19, S23, S25-S29, S39-S40, S43, S45	15
	Time	S4-S6, S11, S16, S30-S31, S33, S36, S40-S41, S44	12
	Threads	S5, S6, S8, S10, S13, S15-S17, S24, S33, S36, S40	12
	Data shared between computational units	S5, S8-S9, S13, S15-S17, S35, S42, S46	10
Source code	Line of code	S3, S5-S6, S10, S17, S20, S22, S24, S29, S33, S45-S46	12
	Class	S1-S2, S7, S11, S14, S18, S21, S28, S36, S40	10
	Structural component	S1, S10-S11, S36	4
Version control system	Code changes	S4	1

Program execution. This category involves the articles that collect or calculate data from program execution. This information facilitates the understanding of the behavior of a program. All the articles extract various data from program execution to support developers with memory consumption analysis. However, we noticed that the information selected varies in different aspects.

During the program execution, a large number of memory events occur. Articles regularly mention that memory traces are collected. However, the concept of memory trace could be too general, so we focused on the details of the memory traces collected. We considered three memory events: (i) data is allocated in sections of memory (allocations), (ii) data allocated is used (read and write), and (iii) the occupied memory that is not needed anymore should be released (deallocation).

- *Memory allocations.* We detected that 26.08% of the articles describe extracting data related to memory allocations but do not consider when the memory is released due to the difficulty of extracting this kind of information [72]. These articles usually provide visualizations that display the objects allocated during the program execution to identify objects that consume more memory and distinguish how these objects are related. However, according to our previous research [9], this information could be insufficient at helping developers detect optimization opportunities and address memory issues quickly.
- *Memory allocations and release.* We found that 28.26% of the articles determine gathering data from memory allocations and memory release by tracking specific instructions (*e.g.,* free, delete) or based on garbage collection events. For example, articles S1 [101], and S2 [102] abstract the heap memory evolution through time to assist developers in quickly detecting memory issues (*e.g.,* memory leaks, memory bloats).

- *Memory accesses.* A total of 45.65% articles collect data from all memory events described previously. Some of these articles present visualizations to support students or developers with understanding memory consumption. For example, article S6 [11] proposes *Memoro*, a profiler with a visualization that shows how a program uses the memory. *Memoro* calculates useful metrics (lifetime, usage, useful lifetime) based on the data extracted (*e.g.,* number of reads, number of writes) from the memory accesses. These defined metrics allow developers to detect inefficient use of memory quickly. On the other hand, other articles propose visualizations for specific aspects, such as cache performance analysis or memory analysis in HPC applications.

Furthermore, articles usually specify collecting metrics (*e.g.,* memory address, size) involved with each memory event. Additionally, some articles describe gathering additional information described below to assist developers with memory consumption analysis.

- *Relationships between functions/methods.* Extracting the calling relationships is a common strategy to assist developers with control-flow analysis [44]. Commercial tools (*e.g.,* JProfiler[4], Yourkit[5]) display this information using a tree structure as shown in Figure 1.

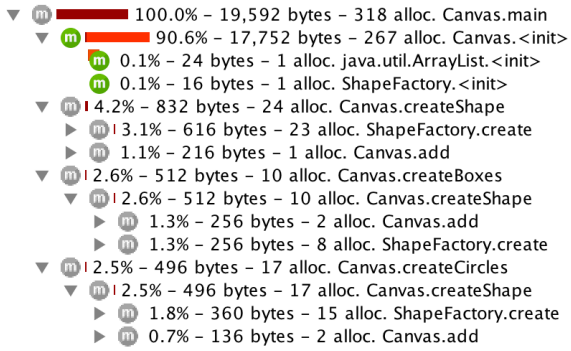

Fig. 1. JProfiler displaying the methods executed with a Calling Context Tree.

We found that 17.39% of the articles describe collecting this information to determine how functions are related to memory events and track specific functions. To exemplify, article **S11** [31] helps developers understand memory consumption by extracting the memory accesses and the call stack. The visualization connects a dense scatter plot for memory accesses and a flame graph for the call stack. As a result, this visualization allows developers to explore through the memory accesses and determine which functions are involved.

- *References between variables.* Some articles proposed visualizations to support developers with data flow analysis. For this reason, 32.60% of the articles specify the extraction of references between variables. The articles focused on analyzing the memory consumption in object-oriented programming languages, which usually display the allocated objects and the references between these objects. For instance, article **S2** [102] proposes *Memory cities*, a visualization to inspect memory growth and reference patterns over objects. Weninger and colleagues employed *Memory cities* to identify memory leaks by examining reference patterns.
- *Time.* We found that 26.08% of the articles explicitly describe collecting how much time is spent executing some instruction or when a memory event occurs to facilitate the program understanding.

[4] https://www.ej-technologies.com/products/jprofiler/overview.html
[5] https://www.yourkit.com

- *Threads.* For 26.08% of the articles specify extracting which threads are involved in memory events. Article **S33** [72] describe that showing the threads created, destroyed and what each thread is doing is fundamental to show programmers detailed information about the program's behavior.
- *Data shared between computational units.* We detect that 21.73% of the articles describe gathering information related to how memory resources are shared among processors. These articles present visualizations to assist developers in understanding the memory management between multiple processors.

Source code. The articles that present static aspects, which are inferred without executing the program belong to this category. We found that 47.82% of articles usually extract static information to help developers map data collected from program execution to source code. The latter benefits developers by identifying and proposing changes on the source code that reduce memory consumption or repair memory issues.

- *Line of code.* We detected that 26.08% of articles extract the file and line of code corresponding to memory events. These articles usually propose visualizations with interaction mechanisms to provide the line of code or a highlighted piece of source code for relating the data from program execution to source code quickly.
  Class. We found that 21.74% of articles collect information at the level of class. As a result, developers can pinpoint classes with specific issues. For example, article **S18** [98] highlights classes that contain methods involved with several allocations or several deallocations.
- *Structural component.* We found that 8.69% of articles gather information about which package or module is involved with memory events. These articles usually present visualizations that group visual elements based on a structural component. For instance, article **S11** [31] displays the functions executed through visual elements and assigns the color of these visual elements based on the module.

Version control systems. Only article **S4** [77] describes extracting data from commits or changes on source code between versions. This article proposes a graph of glyphs to identify memory regressions between consecutive commits.

3.4 RQ3: Representation

This section covers the analysis and categorization of the selected papers based on three relevant aspects in the software visualization field: visual techniques, interactions, and medium.

Table 12. Classification of articles based on the visual technique

Visual techniques	References	Total
Geometrically-transformed	S1-S3, S5, S7, S10, S14, S16-S20, S22-S23, S25-S27, S29, S33, S39, S40, S43, S45	23
Iconic	S2, S4-S5, S7, S14, S22, S25, S28, S33, S36, S40, S43	12
Dense Pixel	S8, S11, S13, S15, S24, S30, S31-S32, S35, S37-S38, S41	12
Stacked	S2, S6, S9, S11, S12, S21, S28, S29, S33, S36, S40	11
Standard 2D/3D	S1, S5-S6, S9-S10, S14-S15, S34, S42, S44, S46	11

3.4.1 Visual techniques. Authors proposing a software visualization employ different visual techniques to explore the information collected from a software application. We categorize the articles according to the classification scheme proposed by Keim [38]. As a result, Table 12 illustrates the distribution of articles based on five categories. In the following, we describe the results of these categories.

- **Geometrically transformed.** According to Keim [38] geometrically transformed techniques transform multi-dimensional data into low dimensional data. This transformation involves mapping an object to a set of points and lines in 2D or 3D (*e.g.,* node-link diagrams, parallel coordinates).
  Half of the selected articles (50%) employ geometrically transformed techniques. This category is the most frequent since several authors propose node-link diagrams to represent relationships, such as object references or the relationships between functions. For instance, article **S26** [1] proposes *Heapviz* to explore and identify primary data structures. *Heapviz* displays a node-link diagram where the nodes represent object instances, and the edges denote the references between objects as shown in Figure 2. *Heapviz* allows developers to identify populated data structures, data structures containing other data structures, and objects referenced by several data structures.

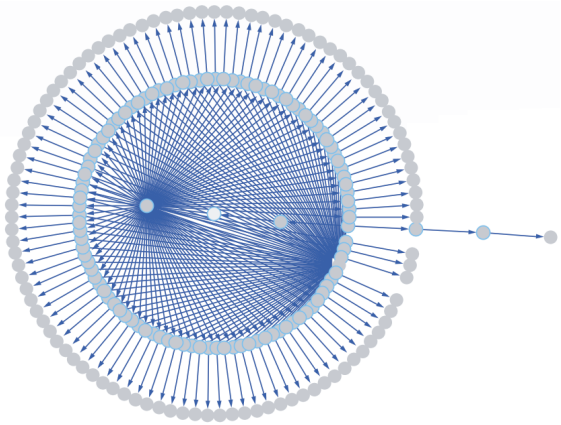

Fig. 2. *Heapviz* [1] presents an interactive visualization to support the analysis of data structures. ©Republished with permission of ACM (Association for Computing Machinery), from "Heapviz: interactive heap visualization for program understanding and debugging", by Edward E. Aftandilian, Sean Kelley, Connor Gramazio, Nathan Ricci, Sara L. Su, and Samuel Z. Guyer. 2010. Permission conveyed through Copyright Clearance Center, Inc.

- **Iconic displays.** This category involves visual techniques, which map the multidimensional data attributes to icon features (*e.g.,* tile bars, star icons). As a result, iconic techniques display icons whose characteristics vary concerning the data attributes.
  Of the selected articles, 26.08% employ iconic techniques. For example, **S4** [77] introduced the glyph called *Spark Circle* to analyze the variations of metrics (objects allocation variation, number of changed methods, execution time variation) between consecutive commits in a commit-graph visualization as shown in Figure 3. In this visualization, each spark circle has three segments, the pink segment for the number of changed methods, the orange segment for the objects allocation variation, and the blue segment for execution time variation. The height of each segment is proportional to the absolute value of the respective metric, and the border is black if any metric increases. As a result, the authors of this visualization detected performance and memory regressions.
- **Dense pixel.** This category includes techniques that represent data values as pixels and group them based on their dimension in specific areas (*e.g.,* matrix visualizations).

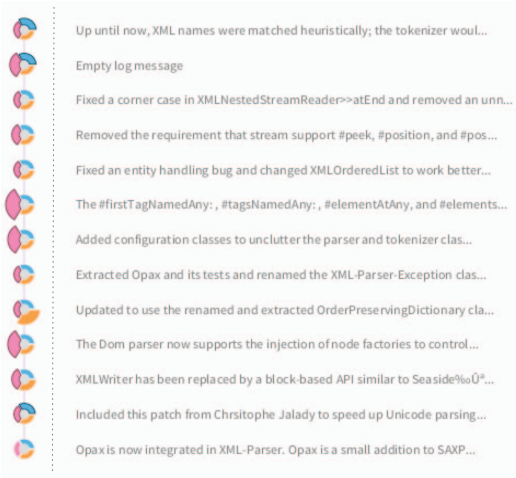

Fig. 3. Visualization proposed by Sandoval [77] to analyze variations between commits. ©2019 Year IEEE. Reprinted, with permission, from "Enhancing Commit Graphs with Visual Runtime Clues" by Juan Pablo Sandoval Alcocer, 2019.

In total, 26.08% of the selected articles use dense pixel techniques to represent a large amount of data (*e.g.*, memory accesses). For example, article **S31** [55] proposes a visualization tool shown in Figure 4 to analyze the behavior of memory allocators in C programs. The main view presents an orthogonal dense pixel layout of time versus memory addresses, which displays hundreds of thousands of allocation events without wasting screen space. The rectangular sizes represent the lifetime and size of blocks, and the color displays the allocation process. This visualization allows developers to analyze memory allocators to optimize their functionality for reducing fragmentation.

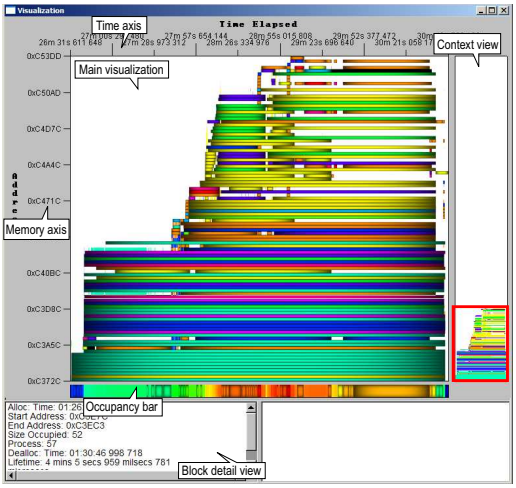

Fig. 4. Visualization proposed by Moreta and Telea [55] to analyze memory allocations behavior. ©2007 Year IEEE. Reprinted, with permission, from "Visualizing Dynamic Memory Allocations" by Sergio Moreta, 2007.

- **Stacked displays.** This category includes visual techniques that show data with a hierarchy structure (*e.g.,* treemaps [82], hierarchical stacking). Of the articles, 23.91% employ stacked displays to represent hierarchical partitioning. To illustrate, Figure 5 shows *Vasco*, an interactive visualization to explore object churn proposed in article **S21** [26]. *Vasco* represents the calling relationships between functions by employing a sunburst. *Vasco* allows users to detect problematical functions by mapping the color and angle to different metrics (*e.g.,* number of allocated objects, number of captured objects). As a result, users can explore functions that allocate many objects that are eventually released and which functions release them. The authors of Vasco demonstrated how to employ their visualization to find and solve memory churn.

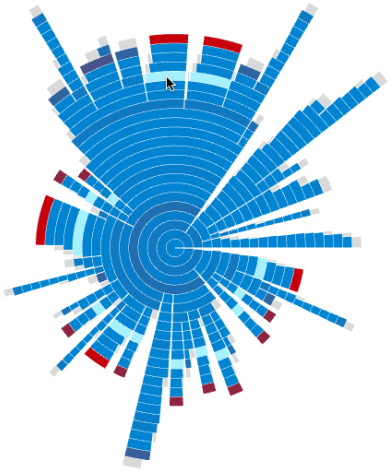

Fig. 5. *Vasco* [26], an interactive visualization to explore object churn. ©2012 Year IEEE. Reprinted, with permission, from "Vasco: A visual approach to explore object churn in framework-intensive applications" by Fleur Duseau, 2012.

- **Standard 2D/3D.** The articles which describe techniques such as plots of two or three dimensions (x-axis, y-axis, and z-axis) belong to this category. In total, 23.91% of the selected papers present standard 2D/3D displays (*e.g.,* bar charts, pie charts). For instance, article **S34** [69] employs standard techniques to analyze cache behavior. *YACO* support developer on understanding cache performance by displaying several bar charts and pie charts to present the statistics on cache hits and misses.

Finally, we found that 39.13% of the selected studies employ more than one visual technique. Consequently, the most popular combination of visual techniques involves the geometrically-transformed display with iconic display.

Views. All the visualizations display the information in one or more views. Commonly, the use of well-integrated multiple views facilitates the exploration of distinct aspects of the data. To illustrate the number of views used on the selected papers, we reviewed the visualization description provided in each one. We found that 47.82% of the studies describe using a single view to display all the information. Most of these papers enrich their visualizations by combining two or more visual techniques, as we described previously. We also noticed that 39.13% of the articles report employing between two to four views. Finally, we detected that 13.04% of the articles adopt more than four views, usually to display other aspects through visualizations with standard 2D/3D techniques.

3.4.2 Interactions. Some visualizations increased their effectiveness by providing interaction options to the users. Usually, a practitioner has the intention of performing some actions over the graphics to facilitate information analysis. However, few articles explicitly describe the supported interactions. This information is usually mixed with the visualization description or a section on usage scenarios. In a number of cases this information was placed in some footnotes. In order to analyze the interaction options that proposed visualizations support, we resorted to classifying only articles that explicitly specify the supported interactions. This classification was based on the taxonomy proposed by Yi and colleagues [107]:

- **Select: mark something as interesting.** Distinguish visual elements of interest is relevant for dense visualizations. We found that 60.86% of the visualizations support this interaction. For example, article **S2** [102] presents *Memory cities*, a visualization to analyze heap evolution using the software city metaphor. In this visualization, the buildings are colored using a gradient ranging from gray to red. *Memory cities* allow the user to highlight a building in blue and thus facilitate its tracking over evolution.
- **Explore: show me something else.** A user can view a limited amount of graphic elements due to a large amount of data and the screen space used to display them. Users usually are interested in seeking out something new by moving the camera across a scene. This category includes interaction techniques (*e.g.*, panning) that allow users to explore different sub-collections of data. We observed that 54.34% of the articles provided exploration techniques. For instance, article **S2** [102] allows moving the camera to view the visualization from above, with a perspective as though the observer were a bird for facilitating inspection of visual elements.
- **Reconfigure: show me a different arrangement.** The arrangement of elements on the screen helps analyze data. We noticed that 17.39% of the visualizations support this task, like article **S7** [9] presents a node-link diagram that allows users to modify the layout by dragging nodes.
- **Encode: show me a different representation.** This category involves the interactions that enable a user to modify the visual representation. We found that 10.86% of the visualizations support metric selection like article **S21** [26] presents *Vasco* that provides a menu bar to change the metrics for color or size of arcs.
- **Abstract/Elaborate: show me more or less detail.** To examine the details of an element of interest is a primary task. Therefore, this category includes details-on-demand interactions. According to Yi and colleagues, the interactions in this category allows developers to adjust the level of abstraction of a data representation. This category is the most frequent in visualizations (60.86%). Usually, the visualizations provide pop-up windows or provide panels with detailed information.
- **Filter: show me something conditionally.** Filtering according to criteria allows users to focus on specific elements quickly. We detected that 32.60% of visualizations enable users to hide elements that do not satisfy a condition. For example, article **S7** [9] presents a menu for excluding methods based on the type of objects that they allocate.
- **Connect: show me related items.** Users focusing on an element of interest will typically explore its relationships with other elements. We found that 21.74% of the visualizations provide interactions to support the navigation through the related elements. For instance, article **S7** [9] facilitate this task by highlighting the edges and nodes related to a selected node.

Additionally, we found that 28.26% of the articles do not explicitly describe the interaction mechanisms provided or specify the intentions of users when they interact with visualizations. We also noticed that the visualization mantra "Overview first, zoom and filter, then details on demand" [83] is not always considered by the proposed visualizations.

3.4.3 Medium. Advanced technology provides users different ways to interact with 3D or 2D visualizations. Maletic and colleagues [46] explained that mediums (*e.g.*, single monitor, wall displays, immersive virtual reality environments) might improve visual representations since they present distinct characteristics. Although monitoring resource consumption may involve some dedicated devices [52], most of the selected visualizations are rendered on a standard monitor of a desktop computer or laptop. Some articles do not explicitly provide the medium, but we inferred that visualizations are rendered on a standard computer screen. As a result, we found that no study exploited the medium to enhance visualizations dedicated to supporting memory consumption analysis tasks.

3.5 RQ4: Evaluation

This section describes the distinct evaluation strategies to validate the effectiveness of the selected software visualizations. We classify the selected studies in three categories based on the work of Merino *et al.* [50]: theoretical, empirical and no explicit evaluation. Table 13 illustrates the distribution of articles based on the strategies used to evaluate visualizations.

Table 13. Classification of articles based on the strategies used to evaluate visualizations

Categories	Strategy	References	Total
Empirical	Usage scenario	S2, S4-S6, S8-S13, S15, S17, S19-S23, S25-S27, S29-S31, S35, S38, S41	26
	Anecdotal evidence	S14, S19, S24, S28, S43	5
	Experiment	S1, S3, S7	3
No explicit evaluation		S16, S18, S32-S34, S36-S37, S39-S40, S42, S44-S46	13

We found that 28.26% of the articles do not provide an evaluation, while the remaining present an empirical evaluation. These empirical evaluations are divided into subcategories described below.

- **Usage scenario.** Of the selected studies, 56.52% only provide application examples. These usage scenarios provide an extended description of how to address memory issues or analyze memory usage with the proposed software visualization. The authors highlight the interactions and the advantages of their visualizations by analyzing popular benchmarks like *DaCapo suite* [8], *DB suite*, *Reptile* [100], *GCOld* [66], *Paraffins*, or open-source projects. Half of the papers in this category presented usage scenarios as case studies. Nonetheless, they do not explicitly describe that professional developers in the industry context with real-world applications employ the visualization. Most of the authors usually give an extended description to demonstrate the effectiveness of their visualization in different cases. However, this description is limited to providing the article authors' experience in using their tool, bearing the risk of biased conclusions according to different articles [106, 109].
- **Anecdotal evidence.** We found that four articles present a short section, usually with the title "industrial experience", where they informally describe the use of the visualization on software companies with professional engineers. In this way, they claim the effectiveness of the visualization, but they do not present data of formal interviews or questionaries. For example, article **S24** [73] collects information from an informal interview to four programmers with the think-aloud method. The goal of this interview is to collect information about how developers employ *Allocray* to detect memory leaks. This research summarizes the usability observations and the feedback of the participants during the interview.

- **Experiment.** Three articles present experiments with participants over software applications. For instance, paper **S7** [9] carries out a user study to evaluate their visualization. The authors explain with details the interviews with eight participants, who use the tool to achieve some tasks. This study describes the results and observations during the work sessions and gathered feedback from the participants.

Finally, none of the selected articles evaluate their visualizations with professional developers and real-world software applications in the context of the software industry. As we mentioned in the category of use scenarios, 28.26% of the articles present sections titled "Case studies", however the users involved in the evaluation are the authors of the articles. According to Merino and colleagues [50], a study that provides an evaluation with authors instead of independent developers, is considered a "Usage Scenario".

3.6 RQ5: Availability

This section lists the selected software visualizations that are available. Most of the software visualizations support a group of people to address a problem of a particular context, this group of people is commonly called *audience* [46]. Having an available tool benefits the target audience to perform software engineering tasks over real-world applications. Also, researchers can conduct how their approach works compared with other visualization proposals by using controlled experiments.

We extracted from the selected articles the tool's name and the links from which the visualization tool is available. However, only 21.73% of the articles provide an existing link, and 6.52% of the articles present a non-existing link. Additionally, we performed web searches based on the article's title, the authors, and the tool's name to find visualization tools available for the remaining articles. As a result, we identified links to visualization tools for 23.91% of the articles due to web search. Table 14 details the information of available software visualizations tools. Table 14 presents the tool's name, where the link was found (article content or web search), the link, and extra information presented aside from the article to install and use the software visualization.

Extra information. We focused on the variety of information that could be available to enrich the experience and facilitate the use of the visualization. Therefore, we detected the presence of the following information:

- *Video.* Some links provide a video explaining features of the visualization. In this case, five links present a video showing the use of the visualization as supplemental material.
- *Sample data.* Some conferences promote the release of datasets to gain public data and the possibility of replicability. Six links provide a list of sample data, which reference the data collected from the applications used in the article as examples.

4 DISCUSSION AND OPEN CHALLENGES

The results described previously provide a general overview of the state-of-art software visualizations centered on the analysis of memory consumption. We have described distinct features and categorized the selected software visualizations to answer our research questions. In this section, we discuss some findings and open challenges for our proposed dimensions. Also, we provide recommendations to practitioners and researchers based on our research questions.

Problem domain. Software visualizations presented in this study attempt to facilitate the analysis and solution of several memory issues (*e.g.*, memory bloat or fragmentation). The design of these visualizations is based on assumptions of developers' needs about the memory issue to be addressed. However, to the best of our knowledge, developer

Table 14. Visualization tools and additional information from the selected articles. Verified on 18/05/2021

ID	Ref.	Tool	Location	Link	Video	Sample Data
S1	[101]	AntTracks	Article	http://mevss.jku.at/?page_id=1592		
S2	[102]	Memory Cities	Article	https://doi.org/10.5281/zenodo.3991785	✓	✓
S3	[35]	PVC	Article	https://github.com/RYOSKATE/PlayVisualizerC.js		✓
S6	[11]	Memoro	Article	https://github.com/epfl-vlsc/memoro		
S7	[9]	–	Article	http://dx.doi.org/10.5281/zenodo.1311787	✓	
S8	[93]	NumaMMA	Article	https://github.com/numamma/numamma		✓
S9	[30]	MemAxes	Web search	https://github.com/LLNL/MemAxes		✓
S13	[24]	Aftermath	Web search	https://www.aftermath-tracing.com/installation/	✓	
S14	[34]	Memory blueprint	Web search	http://smalltalkhub.com/ainfante/MemoryProfiler/		
S15	[6]	Tabarnac	Article	https://github.com/dbeniamine/Tabarnac		
S19	[48]	HeapDbg	Article	http://heapdbg.codeplex.com		
S21	[26]	Vasco	Web search	http://geodes.iro.umontreal.ca/en/projects/vasco/		✓
S26	[1]	Heapviz	Web search	https://github.com/eaftan/heapviz		
S28	[71]	Dyvise	Article	ftp://ftp.cs.brown.edu/u/spr/dyvise.tar.gz		
S31	[55]	–	Web search	http://www.staff.science.uu.nl/~telea001/uploads/Software/MemoView/		
S33	[72]	Jove	Web search	http://cs.brown.edu/~spr/research/visjove.html	✓	
S36	[70]	Jive	Web search	http://cs.brown.edu/~spr/research/vizjive.html	✓	
S37	[68]	GCspy	Web search	https://www.cs.kent.ac.uk/projects/gc/gcspy/		
S39	[111]	Memory graphs	Article	http://www.st.cs.uni-sb.de/memgraphs/		✓
S41	[108]	–	Web search	http://www.cs.toronto.edu/~yijun/cacheviz.guide.html		
S45	[110]	DDD	Web search	https://www.gnu.org/software/ddd/		

design requirements or needs for each particular memory issue has not been fully researched yet. In the past, several studies have analyzed what developers asked during software development [44, 84], revealing a number of needs to be addressed. However, no study provides detailed questions related to memory consumption analysis. Having solid knowledge about developers' needs while addressing these issues may help improve the design and effectiveness of the proposed visualization tools.

Section 3.2 details the classification of articles based on the provided tasks to support programmers over the analysis of memory usage. According to our findings, various visualizations help developers with memory consumption analysis by focusing on different aspects. More than half of the visualizations in *General* and *Data structure* are available. However, few visualizations are available to assist developers in analyzing applications with specific architectures, and one visualization is available for analyzing cache behavior. The unique visualization to analyze memory regression is not available. We also detected that at least two visualizations are available to detect each type of memory issue. However,

the number of visualizations available is reduced. Domain-specific memory analysis, memory issue identification, and memory regression analysis are not fully explored yet, leaving an open opportunity.

Data. Section 3.3 describes the aspects of the software involved in the analysis of memory consumption. According to our findings, a set of articles develop a strategy to gather information, while others use dedicated tools, and the data extracted by these tools are from different projects. The variety of analyzed projects, tools, and data collection strategies makes it difficult to compare proposed visualizations. However, creating a baseline of project set (*i.e.,* projects with particular memory issues) and collection strategies may offer developers and future researchers a guide to successfully gathering specific data and baselines to contrast their tools with state of the art.

Regarding the aspects extracted, we found that most visualizations dismiss mapping the information from program execution with information from source code, like lines of code or classes. Consequently, developers may deal with problems detecting which part of the code is causing or participating in a memory issue. Relating memory metrics with source code is still an open area for further research.

Visual representation. Section 3.4 details the visual techniques used, the interaction options supported, and the medium where the visualization is displayed. We found specific trends in visualizations when using some visual techniques depending on the domain of the problem. For example, most visualizations that assist developers with data structure analysis employ geometrically transformed techniques. However, there is no evidence of the advantages of using a particular visual representation for a single problem domain. In addition, we found that most of the articles present multiple views to display the information. In the same way, we do not observe if using a single view presents better, similar, or worse results than using multiple views.

Finally, we detected that most of the studies employ a single monitor screen to render the visualization. We encourage researchers to analyze the impact of the medium on the effectiveness of visualizations centered on support memory consumption analysis by employing different mediums to render the visualization, like wall-display, multi-touch tables, or a 3D immersive environment.

Evaluation. Section 3.5 summarizes the evaluation strategies used by the selected articles. We found that most articles lack robust empirical evaluation that involves software developers. For instance, we detected that only three articles present evaluations with users of the target audience and expose the comments and observations during the work sessions. We also observed usage scenarios presented as case studies, which detail the author's experience in employing the proposed visualization to analyze memory consumption.

Conducting experiments could be difficult because the nature of the problem domain may require expert developers with a high level of knowledge regarding memory management. Besides, designing and conducting robust experiments is an aspect that memory visualization articles need to improve.

Availability. Actually, only 21.73% of the articles present a valid link where the software visualization tool is available. We detected three articles published between 2002 and 1997 that provide no valid links. Additionally, we found links with software visualization tools for 23.91% of the articles by performing a search on the web that could be tedious, as we explain in Section 3.6. We must highlight that we did not try to install the tool nor verify whether it works. However, we enlisted the additional information (videos that show how to use it correctly or a data sample) that the link presents to support users with the visualization tool.

Unfortunately, around 54.36% of the visualization tools are not available. As a consequence, developers may fail to adopt visualizations to perform tasks related to memory consumption analysis.

5 THREATS OF VALIDITY

Search of articles. A threat to the validity of this study may be not cover all the relevant articles. We performed a systematic search to find articles that propose visualization centered on supporting developers with memory consumption analysis. We developed our search query based on keywords from articles that belong to our scope published between 2017 and 2020 in the most cited venues dedicated to software visualization or memory management. However, our search query is biased by the specific keywords of this set of articles. We decided to decrease this threat by performing an additional manual search and bi-directional snowballing. These two phases assisted in our finding of missing relevant studies.

Selection of articles. A relevant article may be excluded during the selection phase and vice versa. We defined inclusion/exclusion criteria and a quality assessment to reduce bias in selecting articles. During the selection of inclusion/exclusion criteria, the three authors independently review the title and the abstract to consider if an article should be included or not. We calculated the kappa of Fleiss for the inter-rater reliability, and the result was 0,72%, which is generally considered a good agreement. The disagreements were discussed and resolved during meetings among the authors. For the quality assessment, we adopted a checklist to examine the quality of papers. The discrepancies found were reviewed again in a second iteration, and discussion sessions were carried out to reach a consensus.

Data extraction. Another threat to consider is that the data extraction process could be biased. We mitigated this threat by establishing a protocol to extract the data for each paper. An author managed a spreadsheet to keep records of relevant text segments and identify irregularities like missing information. The other two authors of this study review if the data extracted was correct.

Data analysis. During the data analysis, we performed thematic analysis and content analysis to answer our research questions. One author performed a systematic process to conduct a thematic analysis for RQ1 and RQ2. This process includes generating codes and defining themes (patterns) that help answer the research questions. The codes and themes generated vary depending on the coder's experience, point of view, and level of abstraction. For example, to respond to RQ1, we detected visualizations focused on analyzing specific points. However, some articles were too general at determining their objectives, so we decided to consider these articles as a general-purpose group since no specific pattern was found. We tried to reduce this threat by checking the consistency of the process. Due to this, the other two authors examined the description of themes and the data coding. We carried out three discussion meetings to analyze the codes and the themes generated. As a result, we solve the differences among the three authors.

To answer RQ3.1, RQ3.2, and RQ4, we conducted a content analysis. We selected classification schemes proposed in previous studies. We code the data based on these schemes and measure the agreement between the three authors. We detected some specific discrepancies that were discussed and solved during a meeting.

6 RELATED WORK

To the best of our knowledge, this work is the first literature review of software visualizations focused on supporting the user to comprehend memory consumption. Nevertheless, relevant work was published in the software visualization field covering different aspects over the years [64, 65, 74].

Scope. Focus on software visualizations over a general context: these surveys [4, 64, 65, 74, 92], systematic literature reviews [40, 49, 51, 57, 91, 97], taxonomies or classifications [23, 46] generate findings of how visualizations support users on software engineering tasks. In addition, a number of studies cover visualizations supporting specific aspects of the

software engineering field. For example, there are literature reviews [39, 81, 103] that focus on software visualizations to support the analysis software architecture design. These studies analyze how software architecture is visually represented to examine the design based on features like complexity, cohesion, *etc.* Another popular domain is software evolution. For this domain, Novais *et al.* [60] and Salameh *et al.* [76] published systematic studies centered on visualizations to display how certain software elements (*e.g.,* source code, dependencies) change over time.

In addition, several studies [2, 32, 36] focus on the use of software visualization in educational programming. These studies examine the benefits of using visualizations to improve and facilitate the learning process of students. The studies take into account the effectiveness of software visualization to engage students in the field of education. Our study detected six articles that propose software visualizations to help students or novice developers analyze memory consumption.

Furthermore, Bedu *et al.* [5] presented a tertiary systematic literature review on software visualization. This article identifies topic (*e.g.,* architecture, education) trends of surveys focused on software visualizations and issues related to software visualizations (*e.g.,* scalability, validation).

Dimensions. Furthermore, most of the surveys and systematic literature reviews [46, 49, 51, 64, 65, 74] cover the tasks that are supported, the gathered information, and the visual techniques used to display the information. The main variation is the scope of our survey, and consequently, the tasks supported and the collected information are more specific than in prior works. For example, we discussed that our visualizations under study help developers analyze the program behavior and support debugging tasks following the classification scheme of prior work. However, our findings show various focus points (*e.g.,* data structures, cache behavior) to analyze and different memory issues (*e.g.,* memory leak, memory bloat) to address. We also provided which data (*e.g.,* threads, time) and which information sources (*e.g.,* program execution, source code) are collected, similar to the study of Merino *et al.* [51]. However, we considered how the extracted data from different sources is related to help developers with memory consumption analysis.

Furthermore, a minor number of the studies mentioned in this section cover the evaluation and availability dimension. However, there are systematic reviews that focus explicitly on how software visualizations are evaluated. The cases by Merino *et al.* [50], Sensalire *et al.* [79], and Seriai *et al.* [80] examine the different evaluation strategies to validate certain features of a software visualization study (*e.g.,* effectiveness, usability). These studies provide guidelines to produce enough evidence to evaluate software visualizations and describe some challenges in the field. They also explain the weak empirical evidence among software visualizations and detail the inconsistencies in the studies. Our findings expose that 73.91% of the articles present empirical evaluations, mostly usage scenarios. However, the number of studies that describe experiments and case studies is minor. Our results confirm that few articles evaluate visualizations with developers and real-world applications as prior work details.

We noticed that few studies [14, 51] examine the availability of software visualizations. However, our study does not limit publications' data and focuses on visualizations that support memory consumption. Consequently, we provide links to visualization tools not considered by the prior work.

Methodology. As mentioned, most of the relevant prior work focuses on reviewing the state-of-art in the software visualization field. These studies also follow the steps proposed on distinct guidelines for systematic reviews [42, 63], however they present differences with our work in some steps. For example, the construction of the search string could be less complex due to the scope of the studies. Therefore, the number of articles resulting from searching over digital databases and the number of selected papers tends to be higher than ours.

7 CONCLUSION

Our study summarizes the software visualizations to support users with the analysis and improvement of memory consumption. Consequently, we present the supported tasks, data extracted, visual techniques employed, interactions supported, the medium used, evaluations conducted, and a list of visualization tools available.

According to the previous sections, most studies support data structure analysis and memory analysis over applications with specific architectures (*e.g.,* high-performance computing). We detected that visualizations also help developers detect memory issues (*e.g.,* memory bloat, memory leak).

Regarding the data extracted, several articles propose a strategy to gather information or use dedicated tools, such as *Pin* [45], *Jinsight* [20], *DynamoRIO* [6], *etc.* Furthermore, most visualizations dismiss mapping the information from program execution with information from source code, like lines of code or classes. We consider that collecting information from both sources reduces the effort of practitioners to analyze memory consumption.

Additionally, most authors employ more than one visual technique for the software visualizations. We also detected that geometrically transformed display is the most frequent technique because articles propose node-link diagrams to represent relationships between elements. Also, most of the papers use a standard monitor to display the visualization. We consider that visualizations could be implemented to use other mediums such as tactile devices or 3D environments.

Regarding the evaluations conducted, most of the articles present usage scenarios that highlight the visualization features to support users' understanding of memory consumption. Furthermore, most of the applications used in this study are popular benchmarks like *DaCapo suite* [8], *DB suite*, *Reptile* [100], *GCOld* [66], *Paraffins*, or open-source projects. However, only three papers conducted experiments to evaluate the visualization with users. Finally, we detected few visualizations available, and as a consequence, we consider the lack of availability is one of the main weak points in the field.

ACKNOWLEDGMENTS

Alison Fernandez Blanco is supported by a Ph.D. scholarship from CONICYT, Chile. CONICYT-PFCHA/Doctorado Nacional/2019-21191851. Alexandre Bergel is grateful to the ANID FONDECYT Regular project 1200067 for having partially sponsored the work presented in this article.

REFERENCES

[1] Edward E. Aftandilian, Sean Kelley, Connor Gramazio, Nathan Ricci, Sara L. Su, and Samuel Z. Guyer. 2010. Heapviz: Interactive Heap Visualization for Program Understanding and Debugging. In *Proceedings of the 5th International Symposium on Software Visualization* (Salt Lake City, Utah, USA) *(SOFTVIS '10)*. Association for Computing Machinery, New York, NY, USA, 53–62. https://doi.org/10.1145/1879211.1879222

[2] Abdullah Al-Sakkaf, Mazni Omar, and Mazida Ahmad. 2019. A systematic literature review of student engagement in software visualization: a theoretical perspective. *Computer Science Education* 29, 2-3 (2019), 283–309. https://doi.org/10.1080/08993408.2018.1564611

[3] Juan Pablo Sandoval Alcocer, Alexandre Bergel, Stéphane Ducasse, and Marcus Denker. 2013. Performance evolution blueprint: Understanding the impact of software evolution on performance. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 1–9. https://doi.org/10.1109/VISSOFT.2013.6650523

[4] Sarita Bassil and Rudolf K Keller. 2001. Software visualization tools: Survey and analysis. In *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*. IEEE, 7–17. https://doi.org/10.1109/WPC.2001.921708

[5] Laure Bedu, Olivier Tinh, and Fabio Petrillo. 2019. A tertiary systematic literature review on Software Visualization. In *2019 Working Conference on Software Visualization (VISSOFT)*. IEEE, 33–44. https://doi.org/10.1109/VISSOFT.2019.00013

[6] David Beniamine, Matthias Diener, Guillaume Huard, and Philippe O. A. Navaux. 2015. TABARNAC: Visualizing and Resolving Memory Access Issues on NUMA Architectures. In *Proceedings of the 2nd Workshop on Visual Performance Analysis* (Austin, Texas) *(VPA '15)*. Association for Computing Machinery, New York, NY, USA, Article 1, 9 pages. https://doi.org/10.1145/2835238.2835239

[6] http://dynamorio.org

[7] Alexandre Bergel, Felipe Bañados, Romain Robbes, and David Röthlisberger. 2012. Spy: A Flexible Code Profiling Framework. *Comput. Lang. Syst. Struct.* 38, 1 (April 2012), 16–28. https://doi.org/10.1016/j.cl.2011.10.002

[8] Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. 2006. The DaCapo Benchmarks: Java Benchmarking Development and Analysis. *SIGPLAN Not.* 41, 10 (Oct. 2006), 169–190. https://doi.org/10.1145/1167515.1167488

[9] Alison Fernandez Blanco, Juan Pablo Sandoval Alcocer, and Alexandre Bergel. 2018. Effective visualization of object allocation sites. In *2018 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 43–53. https://doi.org/10.1109/VISSOFT.2018.00013

[10] Michael D. Bond and Kathryn S. McKinley. 2008. Tolerating Memory Leaks. *SIGPLAN Not.* 43, 10 (Oct. 2008), 109–126. https://doi.org/10.1145/1449955.1449774

[11] Stuart Byma and James R Larus. 2018. Detailed Heap Profiling. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on Memory Management* (Philadelphia, PA, USA) *(ISMM 2018)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3210563.3210564

[12] Andrew M Cheadle, AJ Field, JW Ayres, Neil Dunn, Richard A Hayden, and J Nystrom-Persson. 2006. Visualising Dynamic Memory Allocators. In *Proceedings of the 5th International Symposium on Memory Management* (Ottawa, Ontario, Canada) *(ISMM '06)*. Association for Computing Machinery, New York, NY, USA, 115–125. https://doi.org/10.1145/1133956.1133972

[13] Adriana E. Chis, Nick Mitchell, Edith Schonberg, Gary Sevitsky, Patrick O'Sullivan, Trevor Parsons, and John Murphy. 2011. Patterns of Memory Inefficiency. In *Proceedings of the 25th European Conference on Object-oriented Programming* (Lancaster, UK) *(ECOOP'11)*. Springer-Verlag, Berlin, Heidelberg, 383–407. https://doi.org/10.1007/978-3-642-22655-7_18

[14] Noptanit Chotisarn, Leonel Merino, Xu Zheng, Supaporn Lonapalawong, Tianye Zhang, Mingliang Xu, and Wei Chen. 2020. A systematic literature review of modern software visualization. *Journal of Visualization* 23, 4 (2020), 539–558. https://doi.org/10.1007/s12650-020-00647-w

[15] ANM Imroz Choudhury, Kristin C Potter, and Steven G Parker. 2008. Interactive visualization for memory reference traces. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 815–822. https://doi.org/10.1111/j.1467-8659.2008.01212.x

[16] ANM Imroz Choudhury and Paul Rosen. 2011. Abstract visualization of runtime memory behavior. In *2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE, 1–8. https://doi.org/10.1109/VISSOF.2011.6069452

[17] ANM Imroz Choudhury, Bei Wang, Paul Rosen, and Valerio Pascucci. 2012. Topological analysis and visualization of cyclical behavior in memory reference traces. In *2012 IEEE Pacific Visualization Symposium*. IEEE, 9–16. https://doi.org/10.1109/PacificVis.2012.6183557

[18] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT press.

[19] Andrew R Dalton and William Kreahling. 2010. Automated Construction of Memory Diagrams for Program Comprehension. In *Proceedings of the 48th Annual Southeast Regional Conference* (Oxford, Mississippi) *(ACM SE '10)*. Association for Computing Machinery, New York, NY, USA, Article 22, 6 pages. https://doi.org/10.1145/1900008.1900040

[20] Wim De Pauw, Erik Jensen, Nick Mitchell, Gary Sevitsky, John Vlissides, and Jeaha Yang. 2002. Visualizing the execution of Java programs. In *Software Visualization*. Springer, 151–162.

[21] Wim De Pauw and Gary Sevitsky. 1999. Visualizing reference patterns for solving memory leaks in Java. In *European Conference on Object-Oriented Programming*. Springer, 116–134. https://doi.org/10.1007/3-540-48743-3_6

[22] Joanna F DeFranco and Phillip A Laplante. 2017. A content analysis process for qualitative software engineering research. *Innovations in Systems and Software Engineering* 13, 2 (2017), 129–141. https://doi.org/10.1007/s11334-017-0287-0

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

[24] Andi Drebes, Antoniu Pop, Karine Heydemann, and Albert Cohen. 2016. Interactive visualization of cross-layer performance anomalies in dynamic task-parallel applications and systems. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 274–283. https://doi.org/10.1109/ISPASS.2016.7482102

[25] Stéphane Ducasse, Michele Lanza, and Roland Bertuli. 2004. High-Level Polymetric Views of Condensed Run-Time Information. In *Proceedings of 8th European Conference on Software Maintenance and Reengineering (CSMR'04)*. IEEE Computer Society Press, Los Alamitos CA, 309–318. https://doi.org/10.1109/CSMR.2004.1281433

[26] Fleur Duseau, Bruno Dufour, and Houari Sahraoui. 2012. Vasco: A visual approach to explore object churn in framework-intensive applications. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 15–24. https://doi.org/10.1109/ICSM.2012.6405248

[27] Satu Elo and Helvi Kyngäs. 2008. The qualitative content analysis process. *Journal of advanced nursing* 62, 1 (2008), 107–115. https://doi.org/10.1111/j.1365-2648.2007.04569.x

[28] Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin* 76, 5 (1971), 378. https://doi.org/10.1037/h0031619

[29] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. 2013. *Statistical methods for rates and proportions*. john wiley & sons.

[30] Alfredo Giménez, Todd Gamblin, Ilir Jusufi, Abhinav Bhatele, Martin Schulz, Peer-Timo Bremer, and Bernd Hamann. 2018. Memaxes: Visualization and analytics for characterizing complex memory performance behaviors. *IEEE transactions on visualization and computer graphics* 24, 7 (2018), 2180–2193. https://doi.org/10.1109/TVCG.2017.2718532

[31] Patrick Gralka, Christoph Schulz, Guido Reina, Daniel Weiskopf, and Thomas Ertl. 2017. Visual exploration of memory traces and call stacks. In *2017 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 54–63. https://doi.org/10.1109/VISSOFT.2017.15

[32] Jeisson Hidalgo-Céspedes, Gabriela Marín-Raventós, and Vladimir Lara-Villagrán. 2016. Learning principles in program visualizations: a systematic literature review. In *2016 IEEE frontiers in education conference (FIE)*. IEEE, 1–9. https://doi.org/10.1109/FIE.2016.7757692

[33] Huihui Nora Huang, Eric Verbeek, Daniel German, Margaret-Anne Storey, and Martin Salois. 2017. Atlantis: Improving the analysis and visualization of large assembly execution traces. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 623–627. https://doi.org/10.1109/ICSME.2017.23

[34] Alejandro Infante and Alexandre Bergel. 2015. Efficiently identifying object production sites. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*. IEEE, 575–579. https://doi.org/10.1109/SANER.2015.7081880

[35] Ryosuke Ishizue, Kazunori Sakamoto, Hironori Washizaki, and Yoshiaki Fukazawa. 2020. PVC. js: visualizing C programs on web browsers for novices. *Heliyon* 6, 4 (2020), e03806. https://doi.org/10.1016/j.heliyon.2020.e03806

[36] Essi Isohanni and Hannu-Matti Järvinen. 2014. Are Visualization Tools Used in Programming Education? By Whom, How, Why, and Why Not?. In *Proceedings of the 14th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '14)*. Association for Computing Machinery, New York, NY, USA, 35–40. https://doi.org/10.1145/2674683.2674688

[37] Kamil Jezek and Richard Lipka. 2017. Antipatterns causing memory bloat: A case study. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 306–315. https://doi.org/10.1109/SANER.2017.7884631

[38] Daniel A Keim. 2002. Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics* 8, 1 (2002), 1–8. https://doi.org/10.1109/2945.981847

[39] Taimur Khan, Henning Barthel, Achim Ebert, and Peter Liggesmeyer. 2012. Visualization and Evolution of Software Architectures. In *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011*, Vol. 27. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 25–42. https://doi.org/10.4230/OASIcs.VLUDS.2011.25

[40] Holger M Kienle and Hausi A Muller. 2007. Requirements of software visualization tools: A literature survey. In *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, 2–9. https://doi.org/10.1109/VISSOF.2007.4290693

[41] Barbara Kitchenham. 2004. Procedures for performing systematic reviews. *Keele, UK, Keele University* 33, 2004 (2004), 1–26.

[42] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE 2007-001. Keele University and Durham University Joint Report. http://www.dur.ac.uk/ebse/resources/Systematic-reviews-5-8.pdf

[43] Michele Lanza and Stéphane Ducasse. 2003. Polymetric Views—A Lightweight Visual Approach to Reverse Engineering. *Transactions on Software Engineering (TSE)* 29, 9 (Sept. 2003), 782–795. https://doi.org/10.1109/TSE.2003.1232284

[44] Thomas D LaToza and Brad A Myers. 2010. Hard-to-answer questions about code. In *Evaluation and Usability of Programming Languages and Tools*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1937117.1937125

[45] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.* 40, 6 (2005), 190–200. https://doi.org/10.1145/1064978.1065034

[46] Jonathan I Maletic, Andrian Marcus, and Michael L Collard. 2002. A task oriented view of software visualization. In *Proceedings First International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, 32–40. https://doi.org/10.1109/VISSOF.2002.1019792

[47] M. Mamani, A. Infante, and A. Bergel. 2014. Inti: Tracking Performance Issue Using a Compact and Effective Visualization. In *2014 33rd International Conference of the Chilean Computer Science Society (SCCC)*. 132–134. https://doi.org/10.1109/SCCC.2014.28

[48] Mark Marron, Cesar Sanchez, Zhendong Su, and Manuel Fahndrich. 2012. Abstracting runtime heaps for program understanding. *IEEE Transactions on Software Engineering* 39, 6 (2012), 774–786. https://doi.org/10.1109/TSE.2012.69

[49] Anna-Liisa Mattila, Petri Ihantola, Terhi Kilamo, Antti Luoto, Mikko Nurminen, and Heli Väätäjä. 2016. Software visualization today: Systematic literature review. In *Proceedings of the 20th International Academic Mindtrek Conference*. Association for Computing Machinery, New York, NY, USA, 262–271. https://doi.org/10.1145/2994310.2994327

[50] Leonel Merino, Mohammad Ghafari, Craig Anslow, and Oscar Nierstrasz. 2018. A systematic literature review of software visualization evaluation. *Journal of systems and software* 144 (2018), 165–180. https://doi.org/10.1016/j.jss.2018.06.027

[51] Leonel Merino, Mohammad Ghafari, and Oscar Nierstrasz. 2016. Towards actionable visualisation in software development. In *2016 IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 61–70. https://doi.org/10.1109/VISSOFT.2016.10

[52] Leonel Merino, Mario Hess, Alexandre Bergel, Oscar Nierstrasz, and Daniel Weiskopf. 2019. PerfVis: Pervasive Visualization in Immersive Augmented Reality for Performance Awareness. In *Companion of the 2019 ACM/SPEC International Conference on Performance Engineering* (Mumbai, India) *(ICPE '19)*. Association for Computing Machinery, New York, NY, USA, 13–16. https://doi.org/10.1145/3302541.3313104

[53] Nick Mitchell, Edith Schonberg, and Gary Sevitsky. 2009. Making sense of large heaps. In *European Conference on Object-Oriented Programming*. Springer-Verlag, Berlin, Heidelberg, 77–97. https://doi.org/10.1007/978-3-642-03013-0_5

[54] Ghanavati Mohammadreza, Diego Costa, Janos Seboek, David Lo, and Artur Andrzejak. 2020. Memory and resource leak defects and their repairs in Java projects. *Empirical Software Engineering* 25, 1 (2020), 678–718. https://doi.org/10.1007/s10664-019-09731-8

[55] Sergio Moreta and Alexandru Telea. 2007. Visualizing dynamic memory allocations. In *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, 31–38. https://doi.org/10.1109/VISSOF.2007.4290697

[56] Tao Mu, Jie Tao, Martin Schulz, and Sally A McKee. 2003. Interactive locality optimization on numa architectures. In *Proceedings of the 2003 ACM Symposium on Software Visualization* (San Diego, California) *(SoftVis '03)*. Association for Computing Machinery, New York, NY, USA, 133–ff. https://doi.org/10.1145/774833.774853

[57] Richard Müller and Dirk Zeckzer. 2015. Past, Present, and Future of 3D Software Visualization. (2015), 63–74. https://doi.org/10.5220/0005325700630074

[58] Colin Myers and David Duke. 2010. A map of the heap: Revealing design abstractions in runtime structures. In *Proceedings of the 5th International Symposium on Software Visualization*. Association for Computing Machinery, New York, NY, USA, 63–72. https://doi.org/10.1145/1879211.1879223

[59] Lucas Leandro Nesi, Samuel Thibault, Luka Stanisic, and Lucas Mello Schnorr. 2019. Visual performance analysis of memory behavior in a task-based runtime on hybrid platforms. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 142–151. https://doi.org/10.1109/CCGRID.2019.00025

[60] Renato Lima Novais, André Torres, Thiago Souto Mendes, Manoel Mendonça, and Nico Zazworka. 2013. Software evolution visualization: A systematic mapping study. *Information and Software Technology* 55, 11 (2013), 1860–1883. https://doi.org/10.1016/j.infsof.2013.05.008

[61] Kristian Nybom, Adnan Ashraf, and Ivan Porres. 2018. A systematic mapping study on API documentation generation approaches. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 462–469. https://doi.org/10.1109/SEAA.2018.00081

[62] Mariano Martinez Peck, Noury Bouraqadi, Marcus Denker, Stéphane Ducasse, and Luc Fabresse. 2010. Visualizing objects and memory usage. In *Smalltalks'2010*. Buenos Ares, Argentina. https://hal.inria.fr/inria-00531510

[63] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

[64] Blaine Price, Ronald Baecker, and Ian Small. 1998. An introduction to software visualization. *Software visualization* (1998), 3–27.

[65] Blaine A Price, Ronald M Baecker, and Ian S Small. 1993. A principled taxonomy of software visualization. *Journal of Visual Languages & Computing* 4, 3 (1993), 211–266. https://doi.org/10.1006/jvlc.1993.1015

[66] Tony Printezis and David Detlefs. 2000. A generational mostly-concurrent garbage collector. In *Proceedings of the 2nd international symposium on Memory management*. Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/362422.362480

[67] Tony Printezis and Alex Garthwaite. 2002. Visualising the Train garbage collector. *SIGPLAN Not.* 38, 2 supplement (June 2002), 50–63. https://doi.org/10.1145/773039.512436

[68] Tony Printezis and Richard Jones. 2002. GCspy: An Adaptable Heap Visualisation Framework. In *Proceedings of the 17th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Seattle, Washington, USA) *(OOPSLA '02)*. Association for Computing Machinery, New York, NY, USA, 343–358. https://doi.org/10.1145/582419.582451

[69] Boris Quaing, Jie Tao, and Wolfgang Karl. 2005. Yaco: A user conducted visualization tool for supporting cache optimization. In *International Conference on High Performance Computing and Communications*. Springer, 694–703. https://doi.org/10.1007/11557654_80

[70] Steven P Reiss. 2003. Visualizing Java in action. In *Proceedings of the 2003 ACM symposium on Software visualization*. ACM, Association for Computing Machinery, New York, NY, USA, 57–ff. https://doi.org/10.1145/774833.774842

[71] Steven P Reiss. 2009. Visualizing the Java heap to detect memory problems. In *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*. IEEE, 73–80. https://doi.org/10.1109/VISSOF.2009.5336418

[72] Steven P Reiss and Manos Renieris. 2005. JOVE: Java as it happens. In *Proceedings of the 2005 ACM symposium on Software visualization*. ACM, Association for Computing Machinery, New York, NY, USA, 115–124. https://doi.org/10.1145/1056018.1056034

[73] George G Robertson, Trishul Chilimbi, and Bongshin Lee. 2010. Allocray: Memory allocation visualization for unmanaged languages. In *Proceedings of the 5th international symposium on Software visualization*. Association for Computing Machinery, New York, NY, USA, 43–52. https://doi.org/10.1145/1879211.1879221

[74] G-C Roman and Kenneth C Cox. 1993. A taxonomy of program visualization systems. *Computer* 26, 12 (1993), 11–24. https://doi.org/10.1109/2.247643

[75] Paul Rosen. 2013. A visual approach to investigating shared and global memory behavior of CUDA kernels. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 161–170. https://doi.org/10.1111/cgf.12103

[76] Hani Bani Salameh, Ayat Ahmad, and Ashraf Aljammal. 2016. Software evolution visualization techniques and methods-a systematic review. In *2016 7th International Conference on Computer Science and Information Technology (CSIT)*. IEEE, 1–6. https://doi.org/10.1109/CSIT.2016.7549475

[77] Juan Pablo Sandoval Alcocer, Harold Camacho Jaimes, Diego Costa, Alexandre Bergel, and Fabian Beck. 2019. Enhancing Commit Graphs with Visual Runtime Clues. In *2019 Working Conference on Software Visualization (VISSOFT)*. 28–32. https://doi.org/10.1109/VISSOFT.2019.00012

[78] Sekhar R Sarukkai and Andrew Beers. 1996. Monitoring data-structure evolution in distributed message-passing programs. In *Proceedings of HICSS-29: 29th Hawaii International Conference on System Sciences*, Vol. 1. IEEE, 310–319. https://doi.org/10.1109/HICSS.1996.495476

[79] Mariam Sensalire, Patrick Ogao, and Alexandru Telea. 2009. Evaluation of software visualization tools: Lessons learned. In *2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, 19–26. https://doi.org/10.1109/VISSOF.2009.5336431

[80] Abderrahmane Seriai, Omar Benomar, Benjamin Cerat, and Houari Sahraoui. 2014. Validation of software visualization tools: A systematic mapping study. In *2014 Second IEEE Working Conference on Software Visualization*. IEEE, 60–69. https://doi.org/10.1109/VISSOFT.2014.19

[81] Mojtaba Shahin, Peng Liang, and Muhammad Ali Babar. 2014. A systematic review of software architecture visualization techniques. *Journal of Systems and Software* 94 (2014), 161–185. https://doi.org/10.1016/j.jss.2014.03.071

[82] Ben Shneiderman. 1992. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on graphics (TOG)* 11, 1 (1992), 92–99. https://doi.org/10.1145/102377.115768

[83] Ben Shneiderman. 2003. The eyes have it: A task by data type taxonomy for information visualizations. In *The craft of information visualization*. Elsevier, 364–371. https://doi.org/10.1016/B978-155860915-0/50046-9

[84] Jonathan Sillito, Gail C Murphy, and Kris De Volder. 2006. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. Association for Computing Machinery, New York, NY, USA, 23–34. https://doi.org/10.1145/1181775.1181779

[85] Rebecca Smith and Scott Rixner. 2016. Leveraging Managed Runtime Systems to Build, Analyze, and Optimize Memory Graphs. *ACM SIGPLAN Notices* 51, 7 (2016), 131–143. https://doi.org/10.1145/2892242.2892253

[86] Amitabh Srivastava and Alan Eustace. 1994. *ATOM: A system for building customized program analysis tools*. Vol. 29. ACM, New York, NY, USA. https://doi.org/10.1145/178243.178260

[87] Jaishankar Sundararaman and Godmar Back. 2008. HDPV: interactive, faithful, in-vivo runtime state visualization for C/C++ and Java. In *Proceedings of the 4th ACM symposium on Software visualization*. Association for Computing Machinery, New York, NY, USA, 47–56. https://doi.org/10.1145/1409720.1409729

[88] Lin Tan, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and Chengxiang Zhai. 2014. Bug characteristics in open source software. *Empirical Software Engineering* 19, 6 (2014), 1665–1705. https://doi.org/10.1007/s10664-013-9258-8

[89] Jie Tao, Wolfgang Karl, and Martin Schulz. 2001. Visualizing the memory access behavior of shared memory applications on NUMA architectures. In *International Conference on Computational Science*. Springer, 861–870. https://doi.org/10.1007/3-540-45718-6_91

[90] Gareth Terry, Nikki Hayfield, Victoria Clarke, and Virginia Braun. 2017. Thematic analysis. *The Sage handbook of qualitative research in psychology* (2017), 17–37.

[91] Alfredo R Teyseyre and Marcelo R Campo. 2009. An overview of 3D software visualization. *IEEE transactions on visualization and computer graphics* 15, 1 (2009), 87–105. https://doi.org/10.1109/TVCG.2008.86

[92] Scott Tilley and Shihong Huang. 2002. Documenting software systems with views iii: towards a task-oriented classification of program visualization techniques. In *Proceedings of the 20th annual international conference on Computer documentation*. Association for Computing Machinery, New York, NY, USA, 226–233. https://doi.org/10.1145/584955.584988

[93] François Trahay, Manuel Selva, Lionel Morel, and Kevin Marquet. 2018. NumaMMA: NUMA memory analyzer. In *Proceedings of the 47th International Conference on Parallel Processing*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3225058.3225094

[94] Edward Tufte and P Graves-Morris. 2014. The visual display of quantitative information.; 1983.

[95] Muhammad Usman, Emilia Mendes, Francila Weidt, and Ricardo Britto. 2014. Effort estimation in agile software development: a systematic literature review. In *Proceedings of the 10th international conference on predictive models in software engineering*. ACM, Association for Computing Machinery, New York, NY, USA, 82–91. https://doi.org/10.1145/2639490.2639503

[96] Eric van der Deijl, Gerco Kanbier, Olivier Temam, and Elana D. Granston. 1997. A cache visualization tool. *Computer* 30, 7 (1997), 71–78. https://doi.org/10.1109/2.596631

[97] Renan Vasconcelos, Marcelo Schots, and Cláudia Werner. 2014. An information visualization feature model for supporting the selection of software visualizations. In *Proceedings of the 22nd International Conference on Program Comprehension*. Association for Computing Machinery, New York, NY, USA, 122–125. https://doi.org/10.1145/2597008.2597796

[98] Raoul L Veroy, Nathan P Ricci, and Samuel Z Guyer. 2013. Visualizing the allocation and death of objects. In *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on*. IEEE, 1–4. https://doi.org/10.1109/VISSOFT.2013.6650538

[99] Anthony J Viera, Joanne M Garrett, et al. 2005. Understanding interobserver agreement: the kappa statistic. *Fam med* 37, 5 (2005), 360–363.

[100] David Wakeling. 1999. Compiling lazy functional programs for the Java Virtual Machine. *Journal of Functional Programming* 9, 6 (1999), 579–603. https://doi.org/10.1017/S0956796899003603

[101] Markus Weninger, Paul Grünbacher, Elias Gander, and Andreas Schörgenhumer. 2020. Evaluating an Interactive Memory Analysis Tool: Findings from a Cognitive Walkthrough and a User Study. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 75 (June 2020), 37 pages. https://doi.org/10.1145/3394977

[102] Markus Weninger, Lukas Makor, and Hanspeter Mössenböck. 2020. Memory Cities: Visualizing Heap Memory Evolution Using the Software City Metaphor. In *2020 Working Conference on Software Visualization (VISSOFT)*. IEEE, 110–121. https://doi.org/10.1109/VISSOFT51673.2020.00017

[103] Joao Werther, Glauco de Figueiredo Carneiro, and Rita Suzana Pitangueira Maciel. [n.d.]. A Systematic Mapping on Visual Solutions to Support the Comprehension of Software Architecture Evolution. ([n. d.]).

[104] Benjamin Weyers, Christian Terboven, Dirk Schmidl, Joachim Herber, Torsten W Kuhlen, Matthias S Müller, and Bernd Hentschel. 2014. Visualization of memory access behavior on hierarchical NUMA architectures. In *2014 First Workshop on Visual Performance Analysis*. IEEE, 42–49. https://doi.org/10.1109/VPA.2014.12

[105] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/2601248.2601268

[106] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[107] Ji Soo Yi, Youn ah Kang, John Stasko, and Julie A Jacko. 2007. Toward a deeper understanding of the role of interaction in information visualization. *IEEE transactions on visualization and computer graphics* 13, 6 (2007), 1224–1231. https://doi.org/10.1109/TVCG.2007.70515

[108] Yijun Yu, Kristof Beyls, and Erik H D'Hollander. 2001. Visualizing the impact of the cache on program execution. In *Proceedings Fifth International Conference on Information Visualisation*. IEEE, 336–341. https://doi.org/10.1109/IV.2001.942079

[109] Marvin V Zelkowitz and Dolores R. Wallace. 1998. Experimental models for validating technology. *Computer* 31, 5 (1998), 23–31.
[110] Andreas Zeller and Dorothea Lütkehaus. 1996. DDD—a free graphical front-end for UNIX debuggers. *ACM Sigplan Notices* 31, 1 (1996), 22–27. https://doi.org/10.1145/249094.249108
[111] Thomas Zimmermann and Andreas Zeller. 2002. Visualizing memory graphs. In *Software Visualization*. Springer Berlin Heidelberg, Berlin, Heidelberg, 191–204. https://doi.org/10.1007/3-540-45875-1_15

A SEARCH STRING FOR DIGITAL LIBRARIES

Table 15 shows the search queries used for the three digital libraries.

Table 15. Search query for the three digital libraries

Digital library	Search query
ACM	*Abstract*: ((software OR program OR application) AND (visualize OR visualization OR visualisation OR visualizations OR visualisations OR visuals OR visual) AND ("memory heap" OR "memory allocation" OR "memory consume" OR "memory consumption" OR "memory usage" OR "memory management" OR "memory issues" OR "memory issue" "memory bloats" OR "memory leaks" OR "memory access" OR "memory address"))
IEEE Xplore	("Abstract": "software" OR "Abstract": "program" OR "Abstract": "application") AND ("Abstract": "visualize" OR "Abstract": "visualization" OR "Abstract": "visualisation" OR "Abstract": "visualizations" OR "Abstract": "visualisations" OR "Abstract": "visuals" OR "Abstract": "visual") AND ("Abstract": "memory heap" OR "Abstract": "memory allocation" OR "Abstract": "memory consume" OR "Abstract": "memory consumption" OR "Abstract": "memory usage" OR "Abstract": "memory management" OR "Abstract": "memory issues" OR "Abstract": "memory bloats" OR "Abstract": "memory leaks" OR "Abstract": "memory access" OR "Abstract": "memory address")
Scopus	ABS ((software OR program OR application) AND (visualize OR visualization OR visualisation OR visualizations OR visualisations OR visuals OR visual) AND ("memory heap" OR "memory allocation" OR "memory consume" OR "memory consumption" OR "memory usage" OR "memory management" OR "memory issues" OR "memory issue" OR "memory bloats" OR "memory leaks" OR "memory access" OR "memory address")) AND (LIMIT-TO (SUBJAREA , "COMP")) AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar")) AND (LIMIT-TO (LANGUAGE , "English"))