Delft University of Technology
Software Engineering Research Group
Technical Report Series

# A Systematic Survey of Program Comprehension through Dynamic Analysis

Bas Cornelissen, Andy Zaidman, Arie van Deursen, Leon Moonen, and Rainer Koschke

Report TUD-SERG-2008-033


TUD-SERG-2008-033

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: Submitted to IEEE Transactions on Software Engineering.

# A Systematic Survey of Program Comprehension through Dynamic Analysis

Bas Cornelissen, *Student Member, IEEE,* Andy Zaidman, *Member, IEEE,*
Arie van Deursen, *Member, IEEE Computer Society,*
Leon Moonen, *Member, IEEE Computer Society,*
and Rainer Koschke, *Member, IEEE*

Manuscript received January 0, 2000; revised January 0, 2000.

Bas Cornelissen, Andy Zaidman and Arie van Deursen are with Delft University of Technology, The Netherlands; Leon Moonen is with Simula Research Laboratory, Norway; and Rainer Koschke is with University of Bremen, Germany.

**Abstract**

Program comprehension is an important activity in software maintenance, as software must be sufficiently understood before it can be properly modified. The study of a program's execution, known as dynamic analysis, has become a common technique in this respect and has received substantial attention from the research community, particularly over the last decade. These efforts have resulted in a large research body of which currently there exists no comprehensive overview.

This paper reports on a systematic literature survey aimed at the identification and structuring of research on program comprehension through dynamic analysis. From a research body consisting of 4,795 articles published in 14 relevant venues between July 1999 and June 2008 and the references therein, we have systematically selected 172 articles and characterized them in terms of four main facets: activity, target, method, and evaluation. The resulting overview offers insight in what constitutes the main contributions of the field, supports the task of identifying gaps and opportunities, and has motivated our discussion of several important research directions that merit additional consideration in the near future.

**Index Terms**

Systematic literature survey, program comprehension, dynamic analysis

## I. Introduction

One of the most important aspects of software maintenance is to understand the software at hand. Understanding a system's inner workings implies studying such artifacts as source code and documentation in order to gain a sufficient level of understanding for a given maintenance task. This *program comprehension* process is known to be very time-consuming, and it is reported that up to 60% of the software engineering effort is spent on understanding the software system at hand [19], [12].

Dynamic analysis, or the analysis of data gathered from a running program, has the potential to provide an accurate picture of a software system because it exposes the system's actual behavior. This picture can range from class-level details up to high-level architectural views [59], [71], [64]. Among the benefits over static analysis are the availability of runtime information and, in the context of object-oriented software, the exposure of object identities and the actual resolution of late binding. A drawback is that dynamic analysis can only provide a partial picture of the system, i.e., the results obtained are valid for the scenarios that were exercised during the analysis.

Dynamic analyses typically comprise the analysis of a system's execution through interpretation (e.g., using the Virtual Machine in Java) or instrumentation, after which the resulting data is used for such purposes as reverse engineering and debugging. Program comprehension constitutes one such purpose, and over the years, numerous dynamic analysis approaches have been proposed in this context, with a broad spectrum of different techniques and tools as a result.

The existence of such a large research body on program comprehension and dynamic analysis necessitates a broad overview of this topic. Through a characterization and structuring of the research efforts to date, existing work can be compared and one can be assisted in such tasks as finding related work and identifying new research opportunities. This has motivated us to conduct a systematic survey of research literature that concerns the use of dynamic analysis in program comprehension contexts.

In order to characterize the articles of interest, we have first performed an exploratory study on the structure of several articles on this topic. This study has led us to decompose typical program comprehension articles into four *facets*:

- The *activity* describes what is being performed or contributed [e.g., view reconstruction or tool surveys].
- The *target* reflects the type of programming language(s) or platform(s) to which the approach is shown to be applicable [e.g., legacy or web-based systems].
- The *method* describes the dynamic analysis methods that are used in conducting the activity [e.g., filtering or concept analysis].
- The *evaluation* outlines the manner(s) in which the approach is validated [e.g., industrial studies or controlled experiments].

Within each facet one can distinguish a series of generic *attributes*: the examples given above (in brackets) are in fact some of the attributes that we use in our framework. With this attribute framework, the papers under study can be characterized in a comprehensive fashion.

The goal of our survey is the systematic selection and characterization of literature that concerns program comprehension through dynamic analysis. Based on the four facets mentioned above, we derive attribute sets to characterize the articles of interest by following a structured approach that involves four main phases and two pilot studies. While our initial focus is on a selection of 14 relevant venues and on the last decade, we include additional literature by following the references therein. The resulting overview offers insight in what constitutes the

main contributions of the field and supports the task of identifying gaps and opportunities. We discuss the implications of our findings and provide recommendations for future work. Specifically, we address the following research questions:

1) Which generic attributes can we identify to characterize the work on program comprehension through dynamic analysis?
2) How is the attention for each of these attributes distributed across the relevant literature?
3) How are each of the main activities typically evaluated?
4) Which recommendations on future directions can we distill from the survey results?

Section II presents an introduction on dynamic analysis for program comprehension. The protocol that lies at the basis of our survey is outlined in Figure 1, which distinguishes four phases that are described in Sections III through VI. Section VII evaluates our approach and findings, and in Section VIII we conclude with a summary of the key contributions of this paper.

## II. Program Comprehension through Dynamic Analysis

To introduce the reader to the field of program comprehension through dynamic analysis, we first provide definitions of *program comprehension* and *dynamic analysis*. The benefits and limitations of dynamic analysis are discussed. We then present a historical overview of the literature in the field, in which we distinguish between early literature and research conducted in the last decade. Finally, we motivate the need to perform a literature survey.

### A. Definitions

Although we intuitively know that we need to understand a software system before being able to maintain it, a general definition of "program comprehension" should prove useful in the context of this survey. The program comprehension definition as introduced by Biggerstaff et al. reflects what constitutes software understanding: *"A person understands a program when he or she is able to explain the program, its structure, its behavior, its effects on its operation context, and its relationships to its application domain in terms that are qualitatively different from the tokens used to construct the source code of the program."* [8] Following this definition, one should understand that `int z = x + y` actually corresponds to the addition of two numbers.

The other central concept of this paper is dynamic analysis, which Ball defines as *"the analysis of the properties of a running software system"* [3]. Note that this definition remains purposely

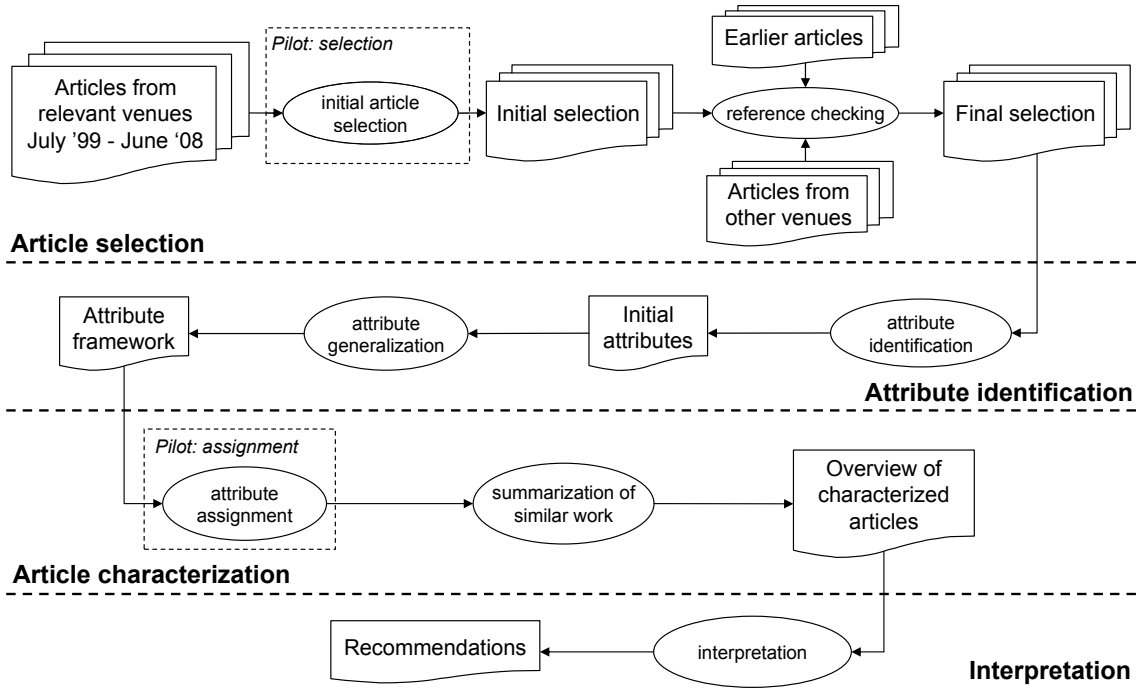


Fig. 1. Overview of the systematic survey process.

vague, as it does not specify which properties are analyzed. To allow the definition to serve in multiple problem domains, the exact properties under analysis are left open.

While the definition of dynamic analysis is rather abstract, we can elaborate upon the benefits and limitations of using dynamic analysis in program comprehension contexts. The benefits that we consider are:

- The *preciseness* with regard to the actual behavior of the software system, for example, in the context of object-oriented software software with its late binding mechanism.
- The fact that a *goal-oriented strategy* can be used, which entails the definition of an execution scenario such that only the parts of interest of the software system are analyzed.

The limitations that we distinguish are:

- The inherent *incompleteness* of dynamic analysis, as the behavior or traces under analysis captures only a small fraction of the usually infinite execution domain of the program under study. Note that the same limitation applies to software testing.
- The difficulty of determining which *scenarios* to execute in order to trigger the program elements of interest. In practice, test suites can be used, or recorded executions involving

user interaction with the system.

- The *scalability* of dynamic analysis due to the large amounts of data that may be introduced in dynamic analysis, affecting performance, storage, and the cognitive load humans can deal with.
- The *observer effect*, i.e., the phenomenon in which software acts differently when under observation, might pose a problem in multithreaded or multi-process software [1], in which timing issues can play a role.

In order to deal with these limitations, many techniques propose abstractions or heuristics, allowing to group program points or execution points that share certain properties. In such cases, a trade-off must be made between recall (are we missing any relevant program points?) and precision (are the program points we direct the user to indeed relevant for his or her comprehension problem?).

### *B. Early research*

From a historical perspective, dynamic analysis was initially used for debugging, testing and profiling. While the purpose of testing is the verification of correctness and while profiling is used to measure (and optimize) performance, debugging is not used to merely locate faults, but also to understand the program at hand.

As programs became larger and more complex, the need to understand software became increasingly important. Originating from the discipline of debugging, the use of dynamic analysis for program comprehension purposes steadily gained more interest. As program comprehension is concerned with conveying (large amounts of) information to humans, the use of *visualization* attracted considerable attention.

Our study of this field showed that the first paper that can be labeled as "program comprehension through dynamic analysis" can be traced back to as early as 1972, when Biermann and Feldman synthesized finite state machines from execution traces [7]. Since then, this type of research has steadily gained momentum, resulting in several important contributions throughout the 1980s and 1990s, which we summarize below.

In 1988, Kleyn and Gingrich [36] proposed structural and behavioral views of object-oriented programs. Their tool, called TraceGraph, used trace information to animate views of program structures.

Five years later, De Pauw et al. [49], [51], [52] started their extensive (and still on-going) research on program visualization, introducing novel views that include matrix visualizations, and the use of "execution pattern" notations to visualize traces in a scalable manner. They were among the first to reconstruct interaction diagrams [30] from running programs, and their work has later resulted in several well-known tools, most notably Jinsight and the associated Eclipse plug-in, TPTP[1].

Wilde and Scully [76] pioneered the field of *feature location* in 1995 with their Software Reconnaissance tool. Feature location concerns the establishment of relations between concepts and source code, and has proven a popular research interest to the present day. Wilde et al. continued the research in this area in the ensuing years with a strong focus on evaluation [75], [74], [73]. At the same time, Lange and Nakamura [41], [40] integrated static and dynamic information to create scalable views of object-oriented software in their Program Explorer tool.

Another visualization was presented by Koskimies and Mössenböck [37] in 1996, involving the reconstruction of scenario diagrams from execution traces. The associated tool, called Scene, offers several abstraction techniques to handle the information overload. Sefika et al. [65] reasoned from a higher level of abstraction in their efforts to generate architecture-oriented visualizations.

In 1997, Jerding et al. [33], [31] proposed their well-known ISVis tool to visualize large execution traces. Two linked views were offered: a continuous sequence diagram, and the "information mural" [32]: a dense, navigable representation of an entire trace.

Walker et al. [71] presented their AVID tool a year later, which visualizes dynamic information at the architectural level. It abstracts the number of runtime objects and their interactions in terms of a user-defined, high-level architectural view (cf. Reflexion [46]).

Finally, in 1999, Ball [3] introduced the concept of frequency spectrum analysis. He showed how the analysis of frequencies of program entities in execution traces can help software engineers decompose programs and identify related computations. In the same year, Richner and Ducasse [59] used static and dynamic information to reconstruct architectural views. They continued this work later on [60], with their focus shifting to the recovery of collaboration diagrams with Prolog queries in their Collaboration Browser tool.

[1]The Eclipse Test & Performance Tools Platform Project, `http://www.eclipse.org/tptp/`

### C. Research in the last decade

Around the turn of the millennium, we witness an increasing research effort in the field of program comprehension through dynamic analysis. The main activities in existing literature were generally continued, i.e., there do not seem to have emerged fundamentally new subfields. Due to the sheer size of the research body of the last decade, we limit ourselves to a selection of notable articles and discuss them in terms of their activities.

As program comprehension is primarily concerned with conveying information to humans, the use of *visualization* techniques is a popular approach that crosscuts several subfields.

One such purpose is *trace analysis*. A popular visualization technique in this respect is the UML sequence diagram, used by (e.g.) De Pauw et al. [50], Systä et al. [70], and Briand et al. [10]. Most of these approaches offer certain measures to address scalability issues, such as metrics and pattern summarization. Popular trace compaction techniques are offered by Reiss and Renieris [58] and by Hamou-Lhadj et al. [26], [24], [25].

From a higher level perspective, there have been several approaches toward *design* and *architecture recovery*. Among these efforts are influential articles by Heuzeroth et al. [28], [27], who combine static and dynamic analyses to detect design patterns in legacy code. Also of interest is the work on architecture reconstruction by Riva [61], [62], and DiscoTect, a tool by Schmerl et al. [79], [64] that constructs state machines from event traces in order to generate architectural views.

Another portion of the research body can be characterized as the study of *behavioral* aspects. The aforementioned work by Heuzeroth et al. analyzes running software by studying interaction patterns. Other notable approaches include a technique by Koskinen et al. [38], who use behavioral profiles to illustrate architecturally significant behavioral rules, and an article by Cook and Du [11] in which thread interactions are exposed in distributed systems. Furthermore, recently there has been considerable effort in the recovery of protocols [55], specifications [43], and grammars [72].

The final subfield that we distinguish is *feature analysis*. While in this context there exist fundamental analyses of program features such as those by Greevy et al. [22], [23] and by Kothari et al. [39], particularly the activity of feature location has become increasingly popular since the aforementioned work by Wilde and Scully [76]. Influential examples include techniques by

Wong et al. [77] (using execution slices), Eisenbarth et al. [17] (using formal concept analysis), Antoniol and Guéhéneuc [2] (through statistical analyses), and Poshyvanyk et al. [53] (using complementary techniques).

### *D. Structuring the field*

The increasing research interest in program comprehension and dynamic analysis has resulted in many techniques and publications, particularly in the last decade. To keep track of past and current developments and to identify future directions, there is need for an overview that structures the existing literature.

Currently, there exist several literature surveys on subfields of the topic at hand. In 2004, Hamou-Lhadj and Lethbridge [24] discussed eight trace exploration tools in terms of three criteria: trace modeling, abstraction level, and size reduction. In the same year, Pacione et al. [47] evaluated five dynamic visualization tools on a series of program comprehension tasks. Greevy's Ph.D. thesis [21] from 2007 summarized several directions within program comprehension, with an emphasis on feature analysis. Also from 2007 is a study by Reiss [57], who described how visualization techniques have evolved from concrete representations of small programs to abstract representations of larger systems.

However, the existing surveys have several characteristics that limit their usability in structuring the entire research body on program comprehension and dynamic analysis. First, they do not constitute a systematic approach because no explicit literature identification strategies and selection criteria are involved, which hinders the reproducibility of the results. Second, the surveys do not utilize common evaluation or characterization criteria, which makes it difficult to structure their collective outcomes. Third, their scopes are rather restricted, and do not represent a broad perspective (i.e., all types of program comprehension activities).

These reasons have inspired us to conduct a systematic literature survey on the use of dynamic analysis for program comprehension. In doing so, we follow a structured process consisting of four phases. Figure 1 shows the tasks involved, which are discussed in the following sections.

## III. Article Selection

This section describes the first phase, which consists of a pilot study, an initial article selection procedure, and a reference checking phase.

### A. Initial article selection

Since program comprehension is a broad subject that has potential overlaps with such fields as debugging, a clear definition of the scope of our survey is required.

**Identification of research.** Search strategies in literature surveys often involve automatic keyword searches (e.g., [4], [15]). However, Brereton et al. [9] recently pointed out that (1) current software engineering digital libraries do not provide good support for the identification of relevant research and the selection of primary studies, and that (2) in comparison to other disciplines, the standard of abstracts in software engineering publications is poor. The former issue exists because in software engineering and computer science, keywords are not consistent across different venues and organizations such as the ACM and the IEEE. Moreover, within the field of program comprehension there is no usable keyword standard that we are aware of.

Similar to Sjøberg et al. [68], we therefore employ an alternative search strategy that involves the manual selection of articles from a series of highly relevant venues.

Given our context, we consider the five journals and nine conferences in Table I to be the most closely related to program comprehension, software engineering, maintenance, and reverse engineering. Our focus is primarily on the period of July 1999 to June 2008; the initial research body thus consists of 4,795 articles that were published at any of the relevant venues as a full paper or a short paper.

**Selection criteria.** Against the background of our research questions, we define two selection criteria in advance that are to be satisfied by the surveyed articles:

1) The article exhibits a profound relation to program comprehension. The author(s) must state program comprehension to be a goal, and the evaluation must demonstrate the purpose of the approach from a program comprehension perspective. This excludes such topics as debugging and performance analysis.
2) The article exhibits a strong focus on dynamic analysis. For this criterion to be satisfied, the article must *utilize* and *evaluate* one or more dynamic analysis techniques, or concern an approach aimed at the support of such techniques (e.g., surveys).

The suitability of the articles is determined on the basis of these selection criteria, i.e., through a manual analysis of the titles, abstracts, keywords, and (if in doubt) conclusions [9]; borderline cases are resolved by discussion amongst the authors.

TABLE I

VENUES INVOLVED IN THE INITIAL ARTICLE SELECTION.

| Type | Acronym | Description | Total no. articles July'99-June'08 |
|---|---|---|---|
| Journal | TSE | IEEE Transactions on Software Engineering | 583 |
| | TOSEM | ACM Transactions on Software Engineering & Methodology | 113 |
| | JSS | Journal on Systems & Software | 965 |
| | JSME | Journal on Software Maintenance & Evolution | 159 |
| | SP&E | Software – Practice & Experience | 586 |
| Conference | ICSE | International Conference on Software Engineering | 429 |
| | ESEC/FSE | European Software Engineering Conference / Symposium on the Foundations of Software Engineering | 240 |
| | FASE | International Conference on Fundamental Approaches to Software Engineering | 198 |
| | ASE | International Conference on Automated Software Engineering | 233 |
| | ICSM | International Conference on Software Maintenance | 413 |
| | WCRE | Working Conference on Reverse Engineering | 254 |
| | IWPC/ICPC | International Workshop/Conference on Program Comprehension | 218 |
| | CSMR | European Conference on Software Maintenance and Reengineering | 270 |
| | SCAM | International Workshop/Working Conference on Source Code Analysis and Manipulation | 134 |

### *B. Selection pilot study*

While the selection criteria being used may be perfectly understandable to the authors of this survey, they could be unclear or ambiguous to others. Following the advice of Kitchenham [35] and Brereton et al. [9], we therefore conduct a pilot study in advance to validate our selection approach against the opinion of domain experts. The outcomes of this study are used to improve the actual article selection procedure that is performed later on.

To conduct the pilot study, the first two authors randomly pre-selected *candidate* articles, i.e., articles from relevant venues and published between July 1999 and June 2008, of which the titles and abstracts loosely suggest that they are relevant for the survey. Note that this selection also includes articles that are beyond the scope of the survey and should be rejected by the raters.[2]

[2]This latter characteristic intentionally makes the task more challenging for the raters.

The domain experts that serve as raters in the pilot are the last three authors of this survey. Since they were involved in neither the article selection procedure nor in the design thereof, they are unbiased subjects with respect to this study.

Each of the subjects is given the task of reading these articles in detail and identifying, on the basis of the selection criteria defined above, the articles that they feel should be included.

The outcomes are then cross-checked with those of the first two authors, who designed the selection procedure. Following these results, any discrepancies are resolved by discussion and the selection criteria are refined when necessary.

**Pilot study results** The results of the pilot study were favorable: out of the 30 article selections performed, 29 yielded the same outcomes as those produced by the selection designers. These figures suggest that our selection criteria are largely unambiguous. The one article that was assessed differently by one of the subjects concerned the field of impact analysis, which, following a discussion on its relation to program comprehension, was considered beyond the scope of this survey.

### *C. Reference checking*

As previously mentioned, the initial focus of this survey is on selected venues in the period of July 1999 to June 2008. To cover articles of interest published before that time or in alternative venues, we (non-recursively) extend the initial selection with relevant articles that have been cited therein, regardless of publication date and venue but taking the selection criteria into account. This procedure minimizes the chance of influential literature being missed, and results in a *final* article selection.

### *D. Article selection results*

The initial selection procedure resulted in 127 relevant articles that were published between July 1999 and June 2008 in any of the 14 venues in Table I. The reference checking yielded another 45 articles (and 17 additional venues), which were subsequently included in the selection. This resulted in a research body that comprises 172 articles. The full listing of these articles is also available online[3]. Figure 2 shows the distribution of all surveyed articles across the venues

[3] `http://swerl.tudelft.nl/bin/view/Main/ProgCompSurvey`

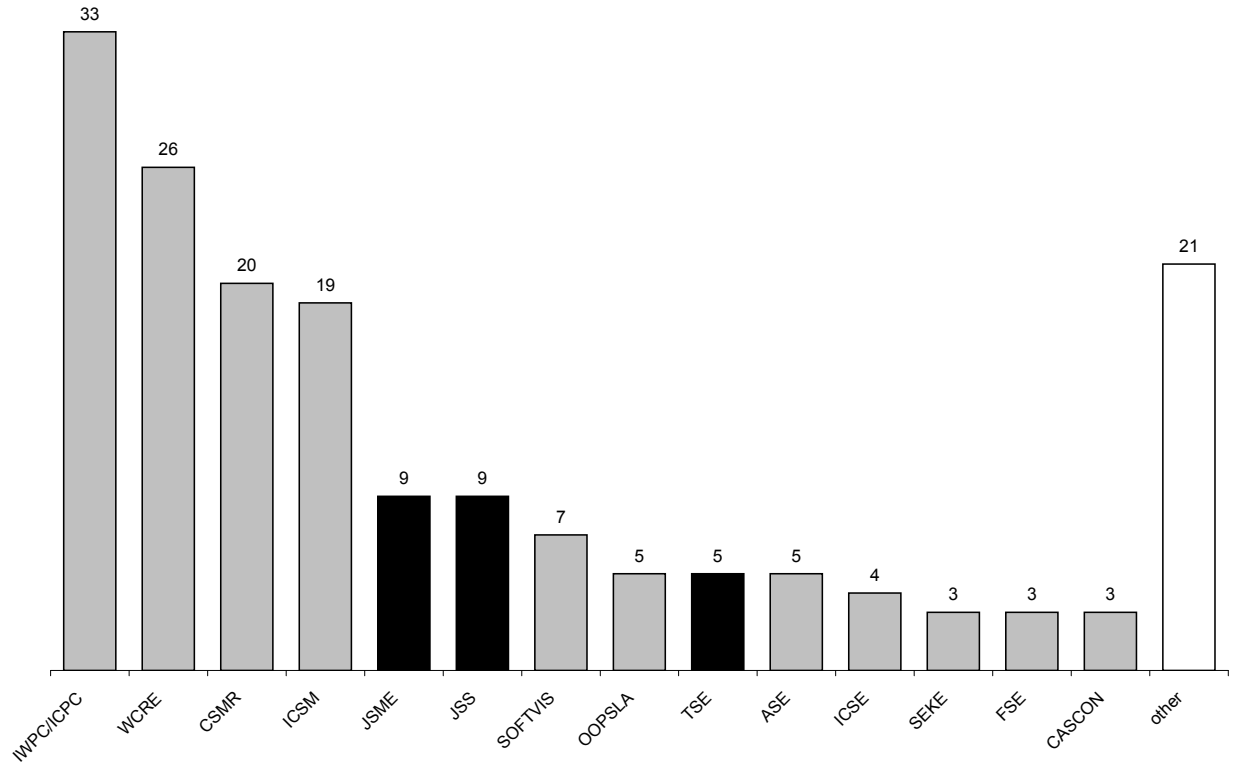


Fig. 2. Distribution of the final article selection across the different venues. Bars in black denote journals; grey bars denote conferences.

from which at least three articles were selected.

## IV. Attribute Framework

As shown in Figure 1, the step after identifying the papers of interest is the construction of an attribute framework that can be used to characterize the selected papers. In this section we describe the process we used to arrive at such a framework, as well as the resulting framework.

### A. Attribute identification

As stated in Section I, our framework distinguishes four facets of interest: the *activity* performed, the type of *target* system analyzed, the *method* developed or used, and the *evaluation* approach used. The goal of our attribute identification step is to refine each of these four facets into a number of specific attributes.

In a first pass, we study all papers, and write down words of interest that could be relevant for a particular facet (e.g., "survey", or "feature analysis" for the activity facet). This data extraction task is performed by the first two authors of this survey. The result after reading all articles is a (large) set of initial attributes.

Note that to reduce the reviewer bias, we do not assume to know any attributes or keywords in advance.

### *B. Attribute generalization*

After the initial attribute sets have been identified, we generalize them in order to render their number manageable and to improve their reusability. This is achieved through a discussion between the first three authors of this survey. Regarding the target facet, for example, the attributes "Java" and "Smalltalk" can intuitively be generalized to "object-oriented languages". After this data synthesis task, the resulting attribute sets are documented.

### *C. Resulting attribute framework*

The use of our attribute framework on the article selection has resulted in seven different activities, six targets, 13 methods, and seven evaluation types. Table II lists the attributes and their descriptions.

The activity facet distinguishes between *five established subfields* within program comprehension: design and architecture recovery, visualization, feature analysis, trace analysis, and behavioral analysis. Each of these five attributes encapsulates a series of closely related activities, of which some were scarcely found: for example, very few authors propose new dynamic slicing techniques[4], and only a handful of articles aim at the reconstruction of state machines for program comprehension. In addition to the five major subfields, we have defined attributes for surveys and general purpose activities. The latter attribute denotes a broad series of miscellaneous activities that are otherwise difficult to generalize, e.g., solutions to the year 2000 problem, new dynamic slicing techniques, or visualizations with no specific focus.

The target facet contains six different types of programming platforms and languages. While we found it interesting to distinguish "legacy" software, this turned out to be difficult in practice,

[4]There exist numerous papers on dynamic slicing, but we found only two that use it in a program comprehension context.

TABLE II

ATTRIBUTE FRAMEWORK.

| Facet | Attribute | Description |
|---|---|---|
| Activity | survey | a survey or comparative evaluation of existing approaches that fulfill a common goal. |
| | design/arch. | the recovery of high-level designs or architectures. |
| | views | the reconstruction of specific views, e.g., UML sequence diagrams. |
| | features | the analysis of features, concepts, or concerns, or relating these to source code. |
| | trace analysis | the understanding or compaction of execution traces. |
| | behavior | the analysis of a system's behavior or communications, e.g., protocol or state machine recovery. |
| | general | gaining a general, non-specific knowledge of a program. |
| Target | legacy | legacy software, if classified as such by the author(s). |
| | procedural | programs written in procedural languages. |
| | oo | programs written in object-oriented languages, with such features as late binding and polymorphism. |
| | threads | multithreaded systems. |
| | web | web applications. |
| | distributed | distributed systems. |
| Method | vis. (std.) | standard, widely used visualization techniques, e.g., graphs or UML. |
| | vis. (adv.) | advanced visualization techniques, e.g, polymetric views or information murals. |
| | slicing | dynamic slicing techniques. |
| | filtering | filtering techniques or selective tracing, e.g., utility filtering. |
| | metrics | the use of metrics. |
| | static | information obtained through static analyses, e.g., from source code or documentation. |
| | patt. det. | algorithms for the detection of design patterns or recurrent patterns. |
| | compr./summ. | compression, summarization, and clustering techniques. |
| | heuristics | the use of heuristics, e.g., probabilistic ranking or sampling. |
| | fca | formal concept analysis. |
| | querying | querying techniques. |
| | online | online analysis, as opposed to post mortem (trace) analysis. |
| | mult. traces | the analysis or comparison of multiple traces. |
| Evaluation | preliminary | evaluations of a preliminary nature, e.g., toy examples. |
| | regular | evaluations on medium-/large-scale open source systems (10K+ LOC) or traces (100K+ events). |
| | industrial | evaluations on industrial systems. |
| | comparison | comparisons of the authors' approach with existing solutions. |
| | human subj. | the involvement of human subjects, i.e., controlled experiments & questionnaires. |
| | quantitative | assessments of quantitative aspects, e.g., speed, recall, or trace reduction rate. |
| | unknown/none | no evaluation, or evaluations on systems of unspecified size or complexity. |

as such a classification depends greatly on one's perspective. For instance, a legacy system could have been written in Fortran or COBOL, lack any documentation, or simply be over 20 years old; on the other hand, it could also be a more modern system that is simply difficult to maintain. Therefore, with respect to the legacy attribute, we rely on the type of the target platform as

formulated by the authors of the papers at hand. Other targets include procedural languages, object-oriented languages, web applications, distributed systems, and software that relies heavily on multithreading.

The method facet is the most versatile of facets, and contains 13 different techniques. Note that we have chosen to distinguish between standard and advanced visualizations: the former denotes ordinary, widely available techniques that are simple in nature, whereas the latter represents more elaborate approaches that are seldomly used (e.g., OpenGL) or simply not publicly available (e.g., information murals [32]). The remaining attributes represent a variety of largely orthogonal techniques that are often used in conjunction with others.

The evaluation facet distinguishes between seven types of evaluations. The "preliminary" attribute refers to early evaluations, e.g., on relatively small programs or traces; in contrast, the "regular" predicate indicates a mature validation that involves (reasonably) large systems or answers actual research questions. Additionally, we have defined an attribute used to express case studies of an industrial nature. Furthermore, comparisons refer to evaluation types in which an approach is compared to existing solutions side-by-side; the involvement of human subjects measures the impact of an approach from a cognitive point of view; and quantitative evaluations are aimed at the assessment of various quantifiable aspects of an approach (e.g., the reduction potential of a trace reduction technique).

## V. Article Characterization

The third phase comprises the assignment of attributes to the surveyed articles, and the use of the assignment results to summarize the research body.

### A. Attribute assignment

Using our attribute framework from the previous section, we process all articles and assign appropriate attribute sets to them. These attributes effectively capture the essence of the articles in terms of the four facets, and allow for a clear distinction between (and comparison of) the articles under study. The assignment process is performed by the first two authors of this survey.

When assigning attributes to an article, we do not consider what the authors claim to contribute, but rather judge for ourselves. For example, papers on sequence diagram reconstruction are not likely to recover high-level architectures; and we consider an approach to target multithreaded

systems if and only if this claim is validated through an evaluation or, at the very least, a plausible discussion.

### B. Summarization of similar work

Certain articles might be extensions to prior work by the same authors. Common examples are journal publications that expand on earlier work published at conferences or workshops, e.g., by providing extra case studies or by employing an additional method, while maintaining the original context. While in our survey all involved articles are studied and characterized, in this report they are summarized to reduce duplication in frequency counts.

We summarize two or more articles (from the same authors) if they concern similar contexts and (largely) similar approaches. This is achieved by assigning the *union* of their attribute subsets to the most recent article, and discarding the other articles at hand. The advantage of this approach is that the number of articles remains manageable at the loss of virtually no information. The listing and characterization of the discarded articles are available in this technical report and on the website mentioned in Section III-D.

### C. Characterization pilot study

As previously mentioned, the attributes were defined and documented by the first two authors of this survey. Since the actual attribute assignment procedure is performed by the same authors, there is a need to verify the quality of the framework because of reviewer bias: the resulting attributes (and by extension, the resulting article characterization) may not be proper and unambiguous. In other words, since the process is subject to interpretation, different reviewers may envision different attribute subsets for one and the same article.

We therefore conduct another pilot study to assess the quality of the attributes and the attribute assignment procedure. The approach is similar to that of the first pilot. From the final article selection, a subset of five articles are randomly picked and given to the domain experts (the last three authors of this survey), along with (an initial version of) the attribute framework in Table II.

The task involves the use of the given framework to characterize each of the five articles. A comparison of the results with those of the first two authors again yields a measure of the

interrater agreement, upon which we discuss any flaws and strengthen the attribute sets and their descriptions.

### *D. Characterization pilot results & implications*

The results of the characterization pilot resulted in generally high agreement on the activity, target, and evaluation facets. Most disagreement occurred for the method facet, which is also the one with the most attributes. This disagreement can be partly attributed to the fact that one rater tried to assign the single most suitable attribute only, whereas the others tried to assign as many attributes as possible. In the ultimate attribute assignments (discussed in the next section), we adopt the latter strategy: for each article, we select *all* attributes that apply to the approach at hand.

In several cases, the action taken upon interrater disagreement was to adjust the corresponding attributes and their descriptions. These adjustments have already been incorporated in Table II.

Specifically, the following measures were taken:

**Activity**

- It was unclear what constitutes a framework, and whether the associated attribute covers contributions such as IDEs. We therefore extended the "framework" attribute to include models, environments, platforms, and architectures.
- At the request of the raters, the "communication" attribute was renamed to "behavior" and extended to include the recovery of state machines.

**Target**

- The raters had difficulty in assigning the "legacy" attribute, as the term "legacy software" is rather vague. For this reason, in each article we decided to rely on the classification of the target platform (legacy or non-legacy) as specified by the author(s).

**Method**

- The "trace comparison" attribute was renamed to "multiple traces", thus broadening its scope as it now covers all techniques that involve two or more traces.
- Whereas formerly there were distinct attributes for compression, merging, and clustering techniques, these are now covered by the newly created "compression/summarization" attribute since they are often hard to distinguish.

- Similarly, the distinct attributes for (a priori) "selective tracing" and (a posteriori) "filtering" were merged to "filtering", as the difference is generally subtle and largely dependent on the manner in which these techniques are described by their authors.
- The description of the "online analysis" attribute was refined to prevent this method from being confused with online approaches in machine learning (often used in state machine recovery), in which "online" denotes so-called active analysis algorithms.

**Evaluation**

- The former "performance" attribute was renamed to "quantitative analysis" to prevent confusion with software performance analysis, and adjusted such that it not merely concerns measurements in terms of speed but also in such terms as recall and precision.
- The "human subjects" attribute was extended to include questionnaires.
- The distinction between open source and industrial studies was strengthened through the addition of the "open source" property to the "regular" attribute.

### *E. Measuring attribute coincidence*

To further evaluate our attribute framework, we analyze the degree to which the attributes in each facet coincide. Against the background of our characterization results, we examine if there are certain attributes that often occur together, and whether such attributes in fact exhibit such an overlap that they should be merged.

We measure this by determining for each attribute how often it coincides with each of the other attributes in that facet. This results in a fraction between 0 and 1 for each attribute combination: 0 if they never coincide, and 1 if each article that has the one attribute also has the other.

### *F. Characterization results*

The characterization and summarization of the 172 selected articles resulted in an overview of 110 articles, shown in Tables IV and V (at the end of this paper). The second column denotes the number of underlying articles (if any) by the same author; the third column indicates whether we could find a reference to a publicly available tool in the article. In rare cases, none of our attributes fitted a certain aspect of an article; in such cases the value for the facet at hand can be considered "other", "unknown", or "none". While the characterization of *all* 172 articles

is available online and in this technical report, henceforth we speak only in terms of the 110 *summarized* articles because they constitute unique contributions.

As previously mentioned, in each article we have focused on its achievements rather than its claims. On several occasions the titles and abstracts have proven quite inaccurate or incomplete in this respect. However, such occasions were not necessarily to the disadvantage of the author(s) at hand: for example, occasionally the related work section is of such quality that it constitutes a respectable survey (e.g., [10], [42]).

The overview in Tables IV and V serves as a useful reference when seeking related work in particular subfields. For example, when looking for literature on trace visualization, one need only identify the articles that have both the "views" and the "trace analysis" attributes. In a similar fashion, one can find existing work on (e.g.) the use of querying and filtering techniques for architecture reconstruction, or learn how fellow researchers have assessed the quantitative aspects of state machine recovery techniques.

Our attribute coincidence measurements yielded no extraordinary results: while certain high fractions were found, none of these merited merges between the attributes involved because these attributes were obviously different in nature.

Table III shows the coincidences for the attribute combinations of which the fraction value exceeds 0.5.

Figure 3 shows for each facet the distribution of the attributes across the summarized articles, which we discuss in the next section.

## VI. Avenues for Future Research

Given the article selection and attribute assignments of Tables IV and V, our final survey step (see Figure 1) consists of interpreting our findings: what patterns can we recognize, what explanations can we offer, which lessons can we learn, and what avenues for further research can we identify? To conduct this step, we analyze the tables, looking for the most and least common attributes, and for interesting attribute combinations. In this section, we offer a selection of the most important outcomes of this analysis.

TABLE III

ATTRIBUTE COINCIDENCE MEASUREMENTS.

| Facet | Attribute #1 | Attribute #2 | Fraction |
|---|---|---|---|
| Activity | general | views | 0.72 |
| | design/arch. | views | 0.71 |
| | survey | views | 0.60 |
| | behavior | views | 0.52 |
| | views | general | 0.50 |
| | trace | views | 0.50 |
| | survey | general | 0.50 |
| Target | threads | oo | 0.68 |
| Method | compr./summ. | vis. (std.) | 0.60 |
| | fca | filtering | 0.50 |
| | fca | mult. traces | 0.60 |
| | fca | static | 0.70 |
| | fca | vis. (std.) | 0.90 |
| | filtering | vis. (std.) | 0.58 |
| | heuristics | metrics | 0.61 |
| | mult. traces | filtering | 0.54 |
| | mult. traces | metrics | 0.57 |
| | online | compr./summ. | 0.62 |
| | online | static | 0.62 |
| | online | vis. (adv.) | 0.62 |
| | patt. det. | vis. (std.) | 0.60 |
| | querying | filtering | 0.54 |
| | querying | static | 0.59 |
| | querying | vis. (std.) | 0.59 |
| | slicing | static | 0.88 |
| | slicing | vis. (std.) | 0.76 |
| | static | vis. (std.) | 0.60 |
| | vis. (adv.) | filtering | 0.54 |
| | vis. (std.) | static | 0.54 |
| Evaluation | comparison | quantitative | 0.82 |
| | comparison | regular | 0.82 |
| | human subj. | regular | 0.62 |
| | industrial | regular | 0.53 |
| | quantitative | regular | 0.95 |

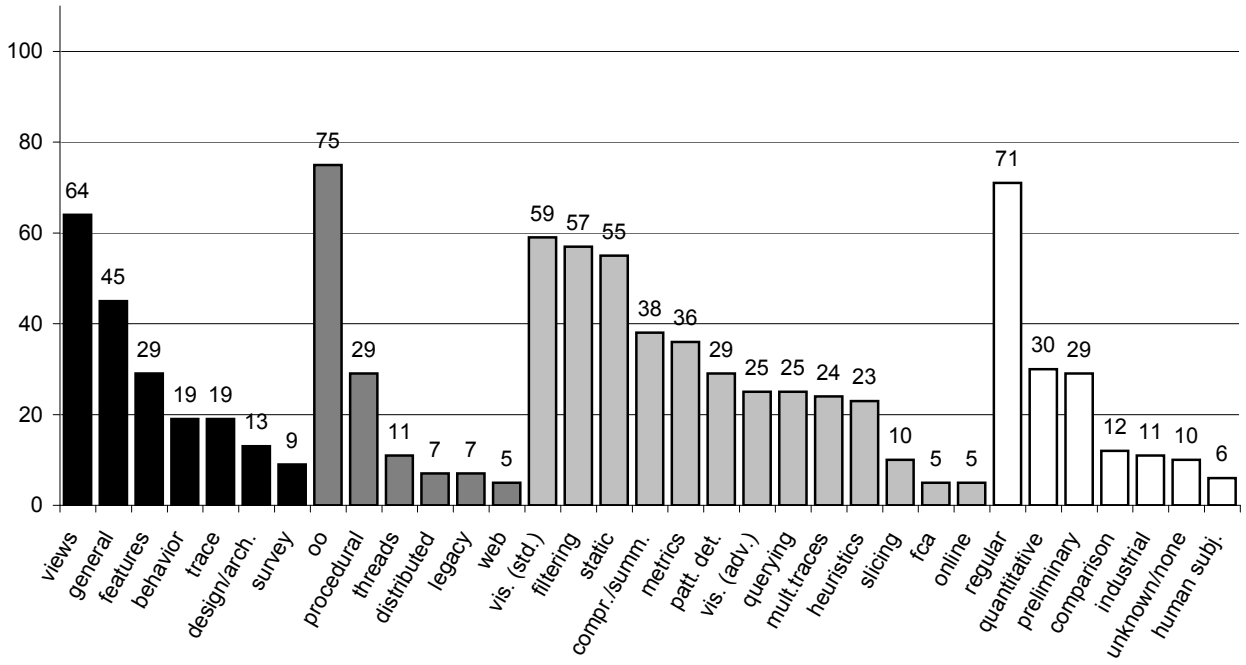


Fig. 3. Distribution of the attributes in each facet across the summarized articles.

### *A. Most common attributes*

Understanding the most common attributes (as displayed in Figure 3) gives an impression of the most widely investigated topics in the field.

Starting with the first facet, the activity, we see that the view attribute is the most common. This is not surprising as program comprehension deals with conveying information to humans, and particularly in the context of dynamic analysis the amounts of information are typically large [81]. We also found many articles to concern general activities, i.e., miscellaneous purposes that could not be generalized to any of the main subfields.

Moving on to the next facet, object-oriented software turns out to be the most common target in the research body: 75 out of the 110 articles propose techniques for, or evaluate techniques on, systems written in (predominantly) Java or Smalltalk. We are not sure why this is the case. Reasons might include ease of instrumentation, the suitability of certain behavioral visualizations (e.g., UML sequence diagrams) for OO systems, the (perceived) complexity of OO applications requiring dynamic analysis, or simply the fact that many program comprehension researchers have a strong interest in object orientation.

Regarding the third facet, the method, we observe that standard visualizations occur more than twice as often as advanced ones. This may have several reasons, among which are the accessibility of standard tools (for graphs, sequence diagrams, and so forth) and possibly the belief that traditional visualizations should suffice in conjunction with efficient abstractions techniques (e.g., filtering). Furthermore, we observe that half of the surveyed articles employ static information. This is in accordance with Ernst's plea for a hybrid approach in which static and dynamic analysis are combined [18].

Finally, within the evaluation facet, we note that regular evaluations (typically using open-source case studies) are the most typical, and that comparisons, industrial case studies, and involvements of human subjects (discussed later on) are rather uncommon. Furthermore, while the assessment of a technique's quantitative aspects is not very commonplace, this evaluation type does appear to be gaining momentum, as more than half (18 out of 30) such evaluations were carried out in the last $2\frac{1}{2}$ years. Interestingly, more than half of these evaluations involved the *feature location* activity; this is further discussed in Section VI-E.

Paraphrasing, one might say that the most popular line of research has been to work on dynamic visualization of open-source object-oriented systems. In the remainder of this section we will look at some of the less popular topics, analyze what the underlying causes for their unpopularity might be, and suggest areas for future research.

### *B. Least common activities*

Within the activity facet, surveys and software architecture reconstruction occurred least.

As discussed in Section II-D, the fact that a satisfactory survey of the field was not available was the starting point for our research, so this did not come as a surprise. Nevertheless, nine papers are labeled as survey, also since we marked papers containing elaborate discussions of related work as surveys (as explained in Section V-F).

In our survey are 13 papers dealing with the use of dynamic analysis to reconstruct software architectures and designs. Some of these papers make use of fairly general traces capturing, e.g., method calls, from which occurrences of design patterns such as the Observer or Mediator can be identified [27].

Another line of research makes use of architecture-aware probing, and aims at visualizing system dynamics in terms of architectural abstractions, such as connectors, locks, dynamically

loaded components, client-server configurations, and so on [65], [64], [29]. While there are not many papers addressing these topics, the initial results do suggest that successful application is possible. We expect that the importance of this field will grow: for complex adaptive systems or dynamically orchestrated compositions of web services, dynamic information may be the only way to understand the runtime architecture.

### *C. Least common targets*

**Web applications.** We were surprised to see that web applications occurred least frequently as target. While traditional web sites consisting of static HTML pages can be easily processed using static analysis alone, modern web applications offer rich functionality for online banking, shopping, e-mailing, document editing, and so on. The logic of these applications is distributed across the browser and the web server, and written using a range of technologies (such as PHP, Javascript, CSS, XSLT, etc.). While this severely complicates static analysis, dynamic analysis might still be possible, for example by monitoring the HTTP traffic between the client and the server.

One complicating factor might be that web applications require user interaction, and hence, user input. Several solutions to this problem exist, such as the use of webserver log files of actual usage, or the use of capture and playback tools for web applications. Furthermore, techniques have been developed to analyze web forms and to fill them in automatically based on a small number of default values provided by the software engineer [5].

The growing popularity of Javascript in general and Ajax (Asynchronous Javascript and XML) in particular, is another argument in favor of dynamic analysis of web applications. With Javascript, events can be programmatically attached to any HTML element. In this setting, even determining the seemingly simple navigation structure of the web application can no longer be done statically, as argued by Mesbah et al. [44]. To deal with this problem, they propose a "crawler" capable of executing Javascript, identifying clickable elements, and triggering clicks automatically: a solution that can also serve as the starting point for dynamic analysis in which client-side logic is to be executed automatically.

**Distributed systems.** As it turns out, the understanding of distributed systems has received little attention in literature: no more than seven articles are concerned with this target type.

Such systems are, however, becoming increasingly popular, e.g., with the advent of service-orientation. Gold et al. paraphrase the core issue as follows: "Service-oriented software lets organizations create new software applications dynamically to meet rapidly changing business needs. As its construction becomes automated, however, software understanding will become more difficult" [20]. Furthermore, distributed systems often behave differently than intended, because of unanticipated usage patterns that are a direct consequence of their dynamic configurability [45]. This increases the need to understand these systems, and due to their heterogeneous nature, dynamic analysis constitutes a viable approach.

**Multithreaded applications.** In recent years, multicore CPUs have become mainstream hardware and multithreading has become increasingly important. The evolution towards multithreaded software is in part evidenced by the foundation of the International Workshop on Multicore Software Engineering (IWMSE), first held at ICSE in 2008: in the proceedings of this workshop it is stated by Pankratius et al. [48] that in the near future "every programmer will be confronted with programming parallel systems", and that in general "parallel components are poorly understood".

The importance of understanding multithreading behavior is not reflected by the current research body: a total of 11 articles are explicitly targeted at multithreaded applications. The use of dynamic analysis on such systems has the important benefit that thread management and interaction can be understood at runtime. A problematic issue in multithreaded systems can be reproducing behavior: does replaying the same scenario result in the same trace? An interesting route to deal with this is to explore the use of multiple traces and suitable trace comparison techniques to highlight essential differences between traces. According to our findings, this is largely unexplored territory: there are only few papers combining the multithreading and trace comparison attributes in our tables.

**Legacy systems** Legacy systems are often in need of a reverse engineering effort, because their internals are poorly understood. Nevertheless, our survey shows that very few papers explicitly mention legacy environments as their target, meaning that dynamic analysis is rarely applied to legacy software systems. This can be partly explained by (1) the fact that researchers do not have access to legacy systems, (2) a lack of available instrumentation tools for legacy platforms, or (3) the fact that instrumented versions of the application are difficult to deploy and, subsequently, run. Another hindering factor is the difficulty of integrating the instrumentation mechanism into

the legacy build process, which is often heterogeneous, i.e., with several kinds of scripting languages in use, and few conventions in place [82].

### *D. Least common evaluations*

**Industrial studies** In our survey we have distinguished between evaluations on industrial and open-source systems. Industrial systems may differ from open-source systems in terms of the way of working, size, complexity, and level of interaction with other systems. Furthermore, industrial systems may share some of the problems of legacy systems as just discussed [14].

We found industrial evaluations to be uncommon, with a total of 11 articles involving industrial cases. Most of these are conducted within the context of research projects with industrial partners, in which the industrial partners have a particular need for reverse engineering.

We have also observed that the degree to which developers or maintainers are involved in the validation is generally low, as their feedback is often limited to answering several general questions, if given at all. This may be a consequence of a lack of time on the part of the developers, or because the industry is not fully aware of the potential benefits of dynamic analysis. This may be resolved by familiarizing practitioners with the benefits, e.g., through the development environment (IDE), as proposed by Röthlisberger et al. [63] who provide dynamic information during programming tasks.

Another impediment for industrial involvement in publications can be fear for disclosing proprietary material. Apart from open discussions with management about the mutual interest, anonymizing traces or presenting aggregated data only might be an option, although obfuscated traces will be even harder to understand.

Finally, a more technical obstacle is the lack of resources, be it memory or processor cycles for the tracing mechanism or disk space for the storage of execution traces. A potential solution to these problems is found in lightweight tracing techniques (e.g., [56]) or capture/replay techniques (e.g., [34], [78]).

**Involvement of human subjects.** In the field of program comprehension, an evaluation that involves human subjects typically seeks to measure such aspects as the usefulness and usability of a tool or technique in practice. The involvement of human subjects is important for program comprehension because this field has the task of conveying information to humans.

Moreover, dynamic analyses are particularly notorious for producing more information than can be comprehended directly [80].

In spite of its importance, this type of evaluation was used in no more than six articles. Bennett et al. [6] use four experts and five graduate students to assess the usefulness of reverse engineered UML sequence diagrams in nine specific comprehension tasks. Quante [54] reports on a controlled experiment with 25 students that involves the use of "object process graphs" in a program comprehension context. Röthlisberger et al. [63] preliminarily assess the added value of dynamic information in an IDE by having six subjects conduct a series of tasks; the authors remain unclear as to the background of the subjects and the nature of the tasks at hand. Hamou-Lhadj and Lethbridge [25] report on a questionnaire in which the quality of a summarized execution trace is judged by nine domain experts; however, no real comprehension tasks are involved. Finally, Wilde et al. [73] and Simmons et al. [67] conduct experiments to assess the practical usefulness of different feature location techniques in legacy Fortran software and in a large 200 kLOC system, respectively.

The design and execution of a controlled experiment is quite elaborate, and requires a great deal of preparation and, preferably, a substantial number of test subjects. Nonetheless, such efforts constitute important contributions to the field of program comprehension and must therefore be encouraged, particularly in case of (novel) visualizations. On a positive note, the fact that three out of the six experiments mentioned above were conducted in 2008 could suggest that this type of evaluation is already gaining momentum.

**Comparisons.** Comparisons (or comparative evaluations) are similar to surveys in the sense that the article at hand involves one or more existing approaches. The difference in terms of our attribute framework is that the authors of side-by-side comparisons do not merely discuss existing solutions, but rather use them *to evaluate their own*. Such a comparison can be more valuable than the evaluation of a technique by itself through anecdotal evidence, as it helps to clarify where there is an improvement over existing work.

Our survey has identified a total of 12 comparative evaluations. The majority of these comparisons was conducted in the context of feature location. As an example, Eaddy et al. [16] discuss two recently proposed feature location techniques, devise one of their own, and subject combinations of the three techniques to a thorough evaluation. Similar approaches are followed

by Antoniol and Guéhéneuc [2] and by Poshyvanyk et al. [53]; in the same context, Wilde et al. [73] offer a comparison between a static and a dynamic technique.

Apart from the field of feature location, in which complementary techniques have already proven to yield the best results, the degree to which existing work is compared against is generally low. One can think of several causes (and solutions) in this context.

First, it must be noted that work on program comprehension cannot always be easily compared because the *human factor* plays an important role. The aforementioned feature location example is an exception, since that activity typically produces quantifiable results; Evaluations of qualitative nature, on the other hand, may require hard to get domain experts or control groups, as well as possibly subjective human interpretation and judgements.

Second, we have determined that only 14 out of the 110 articles offer publicly available *tools*. The lack of available tooling is an important issue, as it hinders the evaluation (and comparison) of the associated approaches by third party researchers. In our earlier work on the assessment of four existing trace reduction techniques [13], for example, we had to resort to our own implementations, which may have resulted in interpretation errors (thus constituting a threat to the internal validity of the experiments). We therefore encourage researchers to make their tools available online, and advocate the use of these tools to compare new solutions against.

Third, the comparison of existing approaches is hindered by the absence of common assessment frameworks and benchmarks, which, as Sim et al. [66] observed, can stimulate technical progress and community building. In the context of program comprehension through dynamic analysis, one could think of using common test sets, such as execution trace repositories (e.g., [13]), and common evaluation criteria, such as the precision and recall measures that are often used in the field of feature location (e.g., [16]). Also of importance in this respect is the use of open-source cases to enable the reproducibility of experiments.

### *E. How activities are evaluated*

In the historical overview in Section II we identified five main subfields in program comprehension: feature analysis, visualization, trace analysis, design and architecture recovery, and behavioral analysis, which correspond to the activity facet of our attribute framework. Here we consider these fields from the perspective of the evaluation facet.

The literature on feature analysis mostly deals with feature location, i.e., relating functionality to source code. What is interesting is not only that this field has received significant attention from 1995 to the present day, but also that comparative evaluations are a common practice in this field, as noticed in Section VI-D. The introduction of common evaluation criteria (i.e., precision and recall) may have contributed to this development. Furthermore, feature analysis accounts for seven out of the 11 industrial evaluations identified in this survey, and for four out of the six evaluations that involve human subjects.

Visualization is a rather different story: for reasons mentioned earlier, the effectiveness of visualization techniques is more difficult to assess, which hinders their comparison and their involvement in industrial contexts. Furthermore, there is still a lot of experimenting going on in this field with both traditional techniques and more advanced solutions. As an example of the former, consider the reverse engineering of UML sequence diagrams: this has been an important topic since the earliest of program comprehension articles (e.g., [49], [37]) and has only recently been subjected to a controlled experiment [6]. In general, the evaluation of visualizations through empirical studies is quite rare, as are industrial studies in this context.

Execution trace analysis, and trace reduction in particular, has received substantial attention in the past decade. This has seldomly resulted in industrial studies and never in controlled experiments. Furthermore, while comparisons with earlier approaches are not very common either, recently there has been a first effort at (quantitatively) evaluating a series of existing reduction techniques side-by-side [13].

Finally, behavioral analysis and architecture recovery are somewhat difficult to assess: the latter has been treated in only five articles, while the former is a rather heterogeneous subfield that comprises various similar, but not equal, disciplines. They are mostly small and involve limited numbers of researchers, and generally these areas of specialization cannot be compared with each other. However, as a behavioral discipline receives more attention in literature, it may grow to become a subfield on its own: the automaton-based recovery of protocols, for example, is a recent development that is adopting common evaluation criteria and thorough comparisons [55], [43].

# VII. Evaluation

In the previous sections we have presented a series of findings based on our paper selection, attribute framework, and attribute assignments. Since conducting a survey is a largely manual task, most threats to validity relate to the possibility of researcher bias, and thus to the concern that other researchers might come to different results and conclusions. One remedy we adopted is to follow, where possible, guidelines on conducting systematic reviews as suggested by, e.g., Kitchenham [35] and Brereton et al. [9]. In particular, we documented and reviewed all steps we made in advance (per pass), including selection criteria and attribute definitions.

In the following sections, we successively describe validity threats pertaining to the article selection, the attribute framework, the article characterization, and the results interpretation, and discuss the manners in which we attempted to minimize their risk.

## *A. Article selection*

Program comprehension is a broad subject that, arguably, has a certain overlap with related topics. Examples of such topics are debugging and impact analysis. The question whether articles of the latter categories should be included in a program comprehension survey is subject to debate. It is our opinion that the topics covered in this survey are most closely related to program comprehension because their common goal is to provide a deeper understanding of the inner workings of software. Following the advice of Kitchenham [35] and Brereton et al. [9], we enforced this criterion by utilizing predefined selection criteria that clearly define the scope, and evaluated these criteria through a pilot study that yielded positive results (Section III-B).

In the process of collecting relevant articles, we chose not to rely on keyword searches. This choice was motivated by a recent paper from Brereton et al. [9], who state that "current software engineering search engines are not designed to support systematic literature reviews"; this observation is confirmed by Staples and Niazi [69]. For this reason, we have followed an alternative search strategy that comprises the manual processing of relevant venues in a certain period of time.

The venues in Table I were chosen because they are most closely related to software engineering, maintenance, and reverse engineering. While this presumption is largely supported by the results (Figure 2), our article selection is not necessarily unbiased or representative of the targeted research body. We have addressed the threat of selection bias by utilizing the aforementioned

selection criteria. Furthermore, we have attempted to increase the representativeness of our selection by following the references in the initial article selection and including the relevant ones in our final selection. We found a non-recursive approach sufficient, as checking for citations within citations typically resulted in no additional articles. As a result, we expect the number of missed articles to be limited; particularly those that have proven influential have almost certainly been included the survey, as they are likely to have been cited often.

### *B. Attribute framework*

We acknowledge that the construction of the attribute framework may be the most subjective step in our approach. The resulting framework may depend on keywords jotted down in the first pass, as well as on the subsequent generalization step. However, the resulting framework can be evaluated in terms of its usefulness: Specifically, we have performed a second pilot study, and measured the degree to which the attributes in each facet coincide. Both of these experiments yielded favorable results and demonstrate the applicability of our framework.

### *C. Article characterization*

Similar to the construction of the attribute framework, the process of applying the framework to the research body is subjective may be difficult to reproduce.

We have addressed this validity threat through a second pilot study (Section V-C), of which the results exposed some discrepancies, mostly within the method facet. The outcomes were discussed among the authors of this survey and resulted in the refinement of several attributes and their descriptions; detailed description of these refinements were provided.

In order to identify topics that have received little attention in the literature, we counted the occurrences of all attributes in the selected articles (shown in Figure 3). A threat to validity in this respect is duplication among articles and experiments: one and the same experiment should not be taken into account twice, which is likely to occur when considering both conference proceedings and journals. We have addressed this threat by summarizing the article selection and using the summarized articles for the interpretation phase.

### *D. Results interpretation*

A potential threat to the validity of the results interpretation is researcher bias, as the interpretation may seek for results that the reviewers were looking for. Our countermeasure has

been a systematic approach towards the analysis of Tables IV and V: in each facet we have discussed the most common and least common attributes. In addition, we have examined the relation between activities and evaluations in particular, as this combination pertains to one of our research questions.

## VIII. Conclusion

In this paper we have reported on a systematic literature survey on program comprehension through dynamic analysis. We have characterized the work on this topic on the basis of four main facets: activity, target, method, and evaluation. While our initial focus was on nine conferences and five journals in the last decade, the use of reference checking to include earlier articles and alternative venues yielded a research body that comprises 31 venues, and relevant articles of up to thirty years old.

Out of 4,795 scanned articles published between July 1999 and June 2008 in 14 relevant venues, we selected the literature that strongly emphasizes the use of dynamic analysis in program comprehension contexts. The addition of relevant articles that were referenced therein resulted in a final selection of 172 articles. Through a detailed reading of this research body, we derived an attribute framework that was consequently used to characterize the articles under study in a structured fashion. The resulting systematic overview is useful as a reference work for researchers in the field of program comprehension through dynamic analysis, and helps them identify both related work and new research opportunities in their respective subfields.

In advance, we posed four research questions pertaining to (1) the identification of generic attributes, (2) the extent to which each of these attributes is represented in the research body, (3) the relation between activities and evaluations, and (4) the distillation of future directions.

The identified attributes are shown in Table II. While being generic in the sense that they characterize all of the surveyed articles, they are sufficiently specific for researchers looking for related work on particular activities, target system types, methods, and evaluation types.

The characterization of the surveyed articles is shown in Tables IV and V. The frequencies of the attributes are provided by Figure 3, which clearly shows the distribution of the attributes in each facet across the research body. We discussed the results, highlighted research aspects that have proven popular throughout the years, and studied the manners in which the major subfields are evaluated.

Based on our analysis of the results, we report on three lessons learned that we feel are the most significant. First, we have observed that the feature location activity sets an example in the way research results are evaluated: this subfield exhibits a great deal of effort in comparing and combining earlier techniques, which has led to a significant technical progress in the past decade (Section VI-E). Second, we conclude that standard object-oriented systems may be overemphasized in the literature, at the cost of web applications, distributed software, and multithreaded systems, for which we have argued that dynamic analysis is very suitable (Section VI-A and Section VI-C). Third, with regard to evaluation, we have learned that comparisons and benchmarking do not occur as often as they should, particularly in activities other than feature location. To support this process, we encourage researchers to make their tools publicly available, and to conduct controlled experiments in case of visualization techniques because these are otherwise difficult to evaluate (Section VI-D).

In summary, the work described in this paper makes the following contributions:

1) A historical overview of the field of program comprehension through dynamic analysis.
2) A selection of key articles in the area of program comprehension and dynamic analysis, based on explicit selection criteria.
3) An attribute framework that can be used to characterize papers in the area of program comprehension through dynamic analysis.
4) An actual characterization of all selected articles in terms of the attributes in this framework.
5) A series of recommendations on future research directions.

## ACKNOWLEDGMENT

Part of this work was conducted at Delft University of Technology in the Reconstructor project, sponsored by NWO/Jacquard.

## REFERENCES

[1] J. Andrews. Testing using log file analysis: tools, methods, and issues. In *Proc. 13th Int. Conf. on Automated Software Engineering (ASE)*, pages 157–166. IEEE Computer Society, 1997.

[2] G. Antoniol and Y.-G. Guéhéneuc. Feature identification: An epidemiological metaphor. *IEEE Trans. Software Eng.*, 32(9):627–641, 2006.

[3] T. Ball. The concept of dynamic analysis. In *Proc. 7th European Software Engineering Conf. & ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE)*, pages 216–234. Springer, 1999.

[4] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp. Motivation in software engineering: A systematic literature review. *Inform. Softw. Technol.*, 50(9-10):860–878, 2008.

[5] M. Benedikt, J. Freire, , and P. Godefroid. VeriWeb: Automatically testing dynamic web sites. In *Proc. 11th Int. Conf. on World Wide Web (WWW)*, 2002.

[6] C. Bennett, D. Myers, D. Ouellet, M.-A. Storey, M. Salois, D. German, and P. Charland. A survey and evaluation of tool features for understanding reverse engineered sequence diagrams. *J. Softw. Maint. Evol.: Res. Pract.*, 20(4):291–315, 2008.

[7] A. W. Biermann. On the inference of turing machines from sample computations. *Artif. Intell.*, 3(1-3):181–198, 1972.

[8] T. J. Biggerstaff, B. G. Mitbander, and D. Webster. The concept assignment problem in program understanding. In *Proc. 15th Int. Conf. on Software Engineering (ICSE)*, pages 482–498. IEEE Computer Society, 1993.

[9] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Software*, 80(4):571–583, 2007.

[10] L. C. Briand, Y. Labiche, and J. Leduc. Toward the reverse engineering of UML sequence diagrams for distributed Java software. *IEEE Trans. Software Eng.*, 32(9):642–663, 2006.

[11] J. E. Cook and Z. Du. Discovering thread interactions in a concurrent system. *J. Syst. Software*, 77(3):285–297, 2005.

[12] T. A. Corbi. Program understanding: Challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306, 1989.

[13] B. Cornelissen, L. Moonen, and A. Zaidman. An assessment methodology for trace reduction techniques. In *Proc. 24th Int. Conf. on Software Maintenance (ICSM)*. IEEE Computer Society, 2008. To appear.

[14] S. Demeyer, S. Ducasse, and O. Nierstrasz. *Object-Oriented Reengineering Patterns*. Morgan Kaufmann, 2003.

[15] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Inform. Softw. Technol.*, 50(9-10):833–859, 2008.

[16] M. Eaddy, A. V. Aho, G. Antoniol, and Y.-G. Guéhéneuc. CERBERUS: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *Proc. 16th Int. Conf. on Program Comprehension (ICPC)*, pages 53–62. IEEE Computer Society, 2008.

[17] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Trans. Software Eng.*, 29(3):210–224, 2003.

[18] M. D. Ernst. Static and dynamic analysis: synergy and duality. In *Proc. 1st ICSE Int. Workshop on Dynamic Analysis (WODA)*, pages 25–28. IEEE Computer Society, 2003.

[19] R. K. Fjeldstad and W. T. Hamlen. Application program maintenance study: Report to our respondents. In *Proc. GUIDE*, volume 48, 1979.

[20] N. Gold, A. Mohan, C. Knight, and M. Munro. Understanding service-oriented software. *IEEE Software*, 21(2):71–77, 2004.

[21] O. Greevy. *Enriching Reverse Engineering with Feature Analysis*. PhD thesis, Universität Bern, 2007.

[22] O. Greevy, S. Ducasse, and T. Gîrba. Analyzing feature traces to incorporate the semantics of change in software evolution analysis. In *Proc. 21st Int. Conf. on Software Maintenance (ICSM)*, pages 347–356. IEEE Computer Society, 2005.

[23] O. Greevy, S. Ducasse, and T. Gîrba. Analyzing software evolution through feature views. *J. Softw. Maint. Evol.: Res. Pract.*, 18(6):425–456, 2006.

[24] A. Hamou-Lhadj and T. C. Lethbridge. A survey of trace exploration tools and techniques. In *Proc. 2004 Conf. of the Centre for Advanced Studies on Collaborative Research (CASCON)*, pages 42–55. IBM Press, 2004.

[25] A. Hamou-Lhadj and T. C. Lethbridge. Summarizing the content of large traces to facilitate the understanding of the

behaviour of a software system. In *Proc. 14th Int. Conf. on Program Comprehension (ICPC)*, pages 181–190. IEEE Computer Society, 2006.

[26] A. Hamou-Lhadj, T. C. Lethbridge, and L. Fu. Challenges and requirements for an effective trace exploration tool. In *Proc. 12th Int. Workshop on Program Comprehension (IWPC)*, pages 70–78. IEEE Computer Society, 2004.

[27] D. Heuzeroth, T. Holl, G. Högström, and W. Löwe. Automatic design pattern detection. In *Proc. 11th Int. Workshop on Program Comprehension (IWPC)*, pages 94–103. IEEE Computer Society, 2003.

[28] D. Heuzeroth, T. Holl, and W. Löwe. Combining static and dynamic analyses to detect interaction patterns. In *Proc. 6th Int. Conf. on Integrated Design and Process Technology (IDPT)*. Society for Design and Process Science, 2002.

[29] T. Israr, M. Woodside, and G. Franks. Interaction tree algorithms to extract effective architecture and layered performance models from traces. *J. Syst. Software*, 80(4):474–492, 2007.

[30] I. Jacobson. *Object-Oriented software engineering: a use case driven approach*. Addison-Wesley, 1995.

[31] D. F. Jerding and S. Rugaber. Using visualization for architectural localization and extraction. In *Proc. 4th Working Conf. on Reverse Engineering (WCRE)*, pages 56–65. IEEE Computer Society, 1997.

[32] D. F. Jerding and J. T. Stasko. The information mural: A technique for displaying and navigating large information spaces. *IEEE Trans. Vis. Comput. Graph.*, 4(3):257–271, 1998.

[33] D. F. Jerding, J. T. Stasko, and T. Ball. Visualizing interactions in program executions. In *Proc. 19th Int. Conf. on Software Engineering (ICSE)*, pages 360–370. ACM, 1997.

[34] S. Joshi and A. Orso. SCARPE: A technique and tool for selective record and replay of program executions. In *Proc. 23rd Int. Conf. on Software Maintenance (ICSM)*, pages 234–243. IEEE Computer Society, 2007.

[35] B. A. Kitchenham. Procedures for performing systematic reviews. In *Technical Report TR/SE-0401, Keele University, and Technical Report 0400011T.1, National ICT Australia*, 2004.

[36] M. F. Kleyn and P. C. Gingrich. Graphtrace - understanding object-oriented systems using concurrently animated views. In *Proc. 3rd Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 191–205. ACM, 1988.

[37] K. Koskimies and H. Mössenböck. Scene: Using scenario diagrams and active text for illustrating object-oriented programs. In *Proc. Proceedings of the 18th Int. Conf. on Software Engineering (ICSE)*, pages 366–375. IEEE Computer Society, 1996.

[38] J. Koskinen, M. Kettunen, and T. Systä. Profile-based approach to support comprehension of software behavior. In *Proc. 14th Int. Conf. on Program Comprehension (ICPC)*, pages 212–224. IEEE Computer Society, 2006.

[39] J. Kothari, T. Denton, A. Shokoufandeh, and S. Mancoridis. Reducing program comprehension effort in evolving software by recognizing feature implementation convergence. In *Proc. 15th Int. Conf. on Program Comprehension (ICPC)*, pages 17–26. IEEE Computer Society, 2007.

[40] D. B. Lange and Y. Nakamura. Interactive visualization of design patterns can help in framework understanding. In *Proc. 10th Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 342–357. ACM, 1995.

[41] D. B. Lange and Y. Nakamura. Program Explorer: A program visualizer for C++. In *Proc. 1st USENIX Conf. on Object-Oriented Technologies and Systems (COOTS)*, pages 39–54. USENIX, 1995.

[42] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich. Feature location via information retrieval based filtering of a single scenario execution trace. In *Proc. 22nd Int. Conf. on Automated Software Engineering (ASE)*, pages 234–243. ACM, 2007.

[43] D. Lo, S. Khoo, and C. Liu. Mining temporal rules for software maintenance. *J. Softw. Maint. Evol.: Res. Pract.*, 20(4):227–247, 2008.

[44] A. Mesbah, E. Bozdag, and A. van Deursen. Crawling Ajax by inferring user interface state changes. In *Proc. 8th Int. Conf. on Web Engineering (ICWE)*, pages 122–134. IEEE Computer Society, 2008.

[45] J. Moe and D. A. Carr. Understanding distributed systems via execution trace data. In *Proc. 9th Int. Workshop on Program Comprehension (IWPC)*, pages 60–67. IEEE Computer Society, 2001.

[46] G. C. Murphy, D. Notkin, and K. J. Sullivan. Software reflexion models: Bridging the gap between design and implementation. *IEEE Trans. Software Eng.*, 27(4):364–380, 2001.

[47] M. J. Pacione, M. Roper, and M. Wood. Comparative evaluation of dynamic visualisation tools. In *Proc. 10th Working Conf. on Reverse Engineering (WCRE)*, pages 80–89. IEEE Computer Society, 2003.

[48] V. Pankratius, C. Schaefer, A. Jannesari, and W. F. Tichy. Software engineering for multicore systems: an experience report. In *Proc. 1st ICSE Int. Workshop on Multicore Software Engineering (IWMSE)*. ACM, 2008.

[49] W. De Pauw, R. Helm, D. Kimelman, and J. M. Vlissides. Visualizing the behavior of object-oriented systems. In *Proc. 8th Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 326–337. ACM, 1993.

[50] W. De Pauw, E. Jensen, N. Mitchell, G. Sevitsky, J. M. Vlissides, and J. Yang. Visualizing the execution of Java programs. In *Proc. ACM 2001 Symp. on Software Visualization (SOFTVIS)*, pages 151–162. ACM, 2001.

[51] W. De Pauw, D. Kimelman, and J. M. Vlissides. Modeling object-oriented program execution. In *Proc. European Object-Oriented Programming Conf. (ECOOP)*, pages 163–182. Springer, 1994.

[52] W. De Pauw, D. Lorenz, J. Vlissides, and M. Wegman. Execution patterns in object-oriented visualization. In *Proc. 4th USENIX Conf. on Object-Oriented Technologies and Systems (COOTS)*, pages 219–234. USENIX, 1998.

[53] D. Poshyvanyk, Y.-G. Guéhéneuc, A. Marcus, G. Antoniol, and V. Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Trans. Software Eng.*, 33(6):420–432, 2007.

[54] J. Quante. Do dynamic object process graphs support program understanding? – a controlled experiment. In *Proc. 16th Int. Conf. on Program Comprehension (ICPC)*, pages 73–82. IEEE Computer Society, 2008.

[55] J. Quante and R. Koschke. Dynamic protocol recovery. In *Proc. 14th Working Conf. on Reverse Engineering (WCRE)*, pages 219–228. IEEE Computer Society, 2007.

[56] S. P. Reiss. Visualizing Java in action. In *Proc. ACM 2003 Symp. on Software Visualization (SOFTVIS)*, pages 57–65. ACM, 2003.

[57] S. P. Reiss. Visual representations of executing programs. *J. Vis. Lang. Comput.*, 18(2):126–148, 2007.

[58] S. P. Reiss and M. Renieris. Encoding program executions. In *Proc. 23rd Int. Conf. on Software Engineering (ICSE)*, pages 221–230. IEEE Computer Society, 2001.

[59] T. Richner and S. Ducasse. Recovering high-level views of object-oriented applications from static and dynamic information. In *Proc. 15th Int. Conf. on Software Maintenance (ICSM)*, pages 13–22. IEEE Computer Society, 1999.

[60] T. Richner and S. Ducasse. Using dynamic information for the iterative recovery of collaborations and roles. In *Proc. 18th Int. Conf. on Software Maintenance (ICSM)*, pages 34–43. IEEE Computer Society, 2002.

[61] C. Riva and J. V. Rodriguez. Combining static and dynamic views for architecture reconstruction. In *Proc. 6th European Conf. on Software Maintenance and Reengineering (CSMR)*, pages 47–55. IEEE Computer Society, 2002.

[62] C. Riva and Y. Yang. Generation of architectural documentation using XML. In *Proc. 9th Working Conf. on Reverse Engineering (WCRE)*, pages 161–179. IEEE Computer Society, 2002.

[63] D. Röthlisberger, O. Greevy, and O. Nierstrasz. Exploiting runtime information in the IDE. In *Proc. 16th Int. Conf. on Program Comprehension (ICPC)*, pages 63–72. IEEE Computer Society, 2008.

[64] B. R. Schmerl, J. Aldrich, D. Garlan, R. Kazman, and H. Yan. Discovering architectures from running systems. *IEEE Trans. Software Eng.*, 32(7):454–466, 2006.

[65] M. Sefika, A. Sane, and R. H. Campbell. Architecture-oriented visualization. In *Proc. 11th Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 389–405. ACM, 1996.

[66] S. E. Sim, S. M. Easterbrook, and R. C. Holt. Using benchmarking to advance research: A challenge to software engineering. In *Proc. 25th Int. Conf. on Software Engineering (ICSE)*, pages 74–83. IEEE Computer Society, 2003.

[67] S. Simmons, D. Edwards, N. Wilde, J. Homan, and M. Groble. Industrial tools for the feature location problem: an exploratory study. *J. Softw. Maint. Evol.: Res. Pract.*, 18(6):457–474, 2006.

[68] D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. *IEEE Trans. Software Eng.*, 31(9):733–753, 2005.

[69] M. Staples and M. Niazi. Experiences using systematic review guidelines. *J. Syst. Software*, 80(9):1425–1437, 2007.

[70] T. Systä, K. Koskimies, and H. A. Müller. Shimba: an environment for reverse engineering Java software systems. *Softw., Pract. Exper.*, 31(4):371–394, 2001.

[71] R. J. Walker, G. C. Murphy, B. N. Freeman-Benson, D. Wright, D. Swanson, and J. Isaak. Visualizing dynamic software system information through high-level models. In *Proc. 13th Conf. on Object-Oriented Programming Systems, Languages & Applications (OOPSLA)*, pages 271–283. ACM, 1998.

[72] N. Walkinshaw, K. Bogdanov, M. Holcombe, and S. Salahuddin. Reverse engineering state machines by interactive grammar inference. *J. Softw. Maint. Evol.: Res. Pract.*, 20(4):269–290, 2008.

[73] N. Wilde, M. Buckellew, H. Page, V. Rajlich, and L. Pounds. A comparison of methods for locating features in legacy software. *J. Syst. Software*, 65(2):105–114, 2003.

[74] N. Wilde, M. Buckellew, H. Page, and Václav Rajlich. A case study of feature location in unstructured legacy fortran code. In *Proc. 5th European Conf. on Software Maintenance and Reengineering (CSMR)*, pages 68–76. IEEE Computer Society, 2001.

[75] N. Wilde and C. Casey. Early field experience with the Software Reconnaissance technique for program comprehension. In *Proc. Int. Conf. on Software Maintenance (ICSM)*, pages 312–318. IEEE Computer Society, 1996.

[76] N. Wilde and M. C. Scully. Software Reconnaissance: Mapping program features to code. *J. Softw. Maint.: Res. Pract.*, 7(1):49–62, 1995.

[77] W. E. Wong, S. S. Gokhale, and J. R. Horgan. Quantifying the closeness between program components and features. *J. Syst. Software*, 54(2):87–98, 2000.

[78] G. Xu, A. Rountev, Y. Tang, and F. Qin. Efficient checkpointing of Java software using context-sensitive capture and replay. In *Proc. 15th European Software Engineering Conf. & ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE)*, pages 85–94. ACM, 2007.

[79] H. Yan, D. Garlan, B. R. Schmerl, J. Aldrich, and R. Kazman. Discotect: A system for discovering architectures from running systems. In *Proc. 26th Int. Conf. on Software Engineering (ICSE)*, pages 470–479. IEEE Computer Society, 2004.

[80] A. Zaidman. *Scalability Solutions for Program Comprehension through Dynamic Analysis*. PhD thesis, University of Antwerp, 2006.

[81] A. Zaidman and S. Demeyer. Managing trace data volume through a heuristical clustering process based on event execution

frequency. In *Proc. 8th European Conf. on Software Maintenance and Reengineering (CSMR)*, pages 329–338. IEEE Computer Society, 2004.

[82] A. Zaidman, S. Demeyer, B. Adams, K. De Schutter, G. Hoffman, and B. De Ruyck. Regaining lost knowledge through dynamic analysis and aspect orientation. In *Proc. 10th European Conf. on Software Maintenance and Reengineering (CSMR)*, pages 91–102. IEEE Computer Society, 2006.

TABLE IV

ARTICLE CHARACTERIZATION RESULTS.

| | | activity | | | | | | | target | | | | | | method | | | | | | | | | | | | | | | | evaluation | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | tool avail. | survey | design/arch. | views | features | trace | behavior | general | legacy | procedural | oo | threads | web | distributed | vis. (std.) | vis. (adv.) | slicing | filtering | metrics | static | patt. det. | compr./summ. | heuristics | fca | querying | online | mult. traces | preliminary | regular | industrial | comparison | human subj. | quantitative | unknown/none |
| Antoniol, 2004 | | . | . | . | . | o | . | . | . | o | o | . | . | . | . | . | . | . | . | . | . | o | o | . | . | . | . | . | o | . | . | . | o | . |
| Antoniol, 2004 | | . | o | o | . | . | . | . | . | . | . | . | o | . | o | . | . | o | . | o | . | . | . | . | . | . | . | . | o | . | . | . | o | . |
| Antoniol, 2005 | | . | . | . | o | . | . | . | . | . | o | o | . | . | . | . | . | o | . | o | . | . | . | . | . | . | o | . | o | . | o | . | o | . |
| Antoniol, 2006 | | o | . | . | o | . | . | . | . | o | o | o | . | . | . | . | . | o | . | o | . | . | . | . | . | . | o | . | o | . | o | . | o | . |
| Ball, 1996 | | . | . | o | . | . | . | o | . | o | . | . | . | . | . | o | o | . | o | o | . | . | . | . | . | . | . | o | . | . | . | . | . | . |
| Ball, 1999 | | . | . | . | . | . | . | o | . | o | . | . | . | . | . | . | . | . | o | o | . | . | . | o | . | . | . | o | . | . | . | . | . | . |
| Bennett, 2008 | | o | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | . | o | o | . | . | . | . | . | . | o | . | . | o | . | . |
| Bojic, 2000 | | . | o | o | . | . | . | . | . | . | o | . | . | . | o | . | . | . | . | . | . | o | . | o | . | . | . | o | . | . | . | . | . | . |
| Briand, 2003 | | o | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . | o | . | . | . | . | . | . |
| Briand, 2005 | | . | . | o | . | . | . | o | . | . | o | o | . | o | o | . | . | . | . | . | o | . | . | . | . | . | . | . | o | . | . | . | o | . |
| Briand, 2006 | o | o | . | o | . | . | . | o | . | . | o | . | . | o | o | . | . | . | . | . | o | . | . | . | . | . | . | . | o | . | . | . | o | . |
| Chan, 2003 | o | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | . | o | o | . | . | . | . | . | o | . | . | . | . | . |
| Cook, 1998 | | . | . | . | . | . | o | . | . | o | . | o | . | . | o | . | . | . | o | . | o | . | o | . | . | . | . | o | . | . | . | . | . | . |
| Cook, 2002 | | . | . | . | . | . | o | . | . | o | . | o | . | . | o | . | . | . | o | . | o | . | o | . | . | . | . | o | . | . | . | . | . | . |
| Cook, 2005 | | . | . | . | . | . | o | . | . | o | . | o | . | . | o | . | . | . | o | . | o | . | o | . | . | . | . | . | o | . | . | . | . | . |
| Cornelissen, 2007 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . |
| Cornelissen, 2007 | o | . | . | o | o | o | . | . | . | . | o | . | . | . | o | o | . | o | . | o | . | o | . | . | . | . | . | . | o | o | . | . | . | . |
| Cornelissen, 2008 | o | . | . | o | o | o | . | o | . | . | o | . | . | . | o | o | . | o | . | o | . | o | . | . | . | . | . | . | o | o | . | . | . | . |
| Cornelissen, 2008 | | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | o | o | . | o | o | o | . | . | . | o | . | o | . | . | . | o | . |
| Dalton, 2008 | | . | . | o | . | . | . | o | . | . | . | . | . | . | o | . | . | o | . | o | . | . | . | . | . | . | . | o | . | . | . | . | o | . |
| Deprez, 2000 | | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | o | o | . | . | . | . | . | . | o | o | . | . | . | . | . | . |
| Ducasse, 2004 | o | . | . | o | . | . | . | o | . | . | o | . | . | . | . | o | . | o | o | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . |
| Eaddy, 2008 | o | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | o | o | . | o | . | . | o | . | o | . | o | . | o | . | o | . | o | . |
| Edwards, 2006 | | . | . | . | o | . | . | . | . | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | o | . | o | o | . | . | . | . |
| Eisenbarth, 2001 | | . | . | o | o | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | o | . | . | o | . | o | . | . | . | . | . |
| Eisenbarth, 2001 | | . | . | o | o | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | o | . | . | o | . | o | . | . | . | . | . |
| Eisenbarth, 2001 | | . | . | o | o | . | . | . | . | o | . | . | . | . | o | . | . | o | . | o | . | . | . | o | . | . | o | . | o | . | . | . | . | . |
| Eisenbarth, 2002 | | . | . | o | o | . | . | . | . | o | . | . | . | . | o | . | . | o | . | o | . | . | . | o | . | . | o | . | o | o | . | . | . | . |
| Eisenbarth, 2003 | | . | . | o | o | . | . | . | . | o | . | . | . | . | o | . | . | o | . | o | . | . | . | o | . | . | o | . | o | o | . | . | . | . |
| Eisenberg, 2005 | | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . | . | o | o | . | . | o | . | o | . | . | . | o | . | o | . | o | . |
| El-Ramly, 2001 | | . | . | . | o | . | o | . | o | . | . | . | . | . | o | . | . | . | . | . | . | o | . | . | . | . | . | . | . | o | o | . | o | . |
| El-Ramly, 2002 | | . | . | . | . | . | o | . | o | . | . | . | . | . | o | . | . | . | . | . | o | o | . | . | . | . | . | . | . | o | . | . | . | . |
| El-Ramly, 2002 | | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . | . | . | . | . | o | o | . | . | . | . | . | . | . | o | . | . | . | . |
| El-Ramly, 2002 | | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . | . | . | o | . | o | o | . | . | . | . | . | . | . | o | . | . | . | . |
| Fischer, 2005 | | . | . | o | . | o | . | . | . | o | . | o | . | . | . | o | . | . | o | . | . | . | . | . | . | . | o | . | o | . | . | . | . | . |
| Fisher II, 2007 | | . | . | . | . | . | o | . | . | . | . | . | o | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | o | . | . | . | o | . |
| Gargiulo, 2001 | o | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . |
| Greevy, 2005 | | . | . | o | o | . | . | . | . | . | o | . | . | . | . | o | . | o | o | . | . | o | . | . | . | . | o | . | o | . | . | . | . | . |
| Greevy, 2006 | | . | . | o | o | . | . | . | . | . | o | . | . | . | . | o | . | o | o | . | . | o | . | . | . | . | o | . | o | . | . | . | . | . |
| Greevy, 2005 | | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . | o | o | o | . | . | . | . | . | . | o | . | o | . | . | . | . | . |
| Greevy, 2005 | | . | . | o | o | . | . | . | . | . | o | . | . | . | . | o | . | o | o | o | . | . | . | o | . | o | . | . | o | . | . | . | . | . |
| Greevy, 2006 | | . | . | o | o | . | . | . | . | . | o | . | . | . | . | o | . | o | o | o | . | . | . | o | . | o | . | . | o | . | . | . | . | . |
| Gschwind, 2003 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . |
| Gschwind, 2003 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . |
| Guéhéneuc, 2002 | o | . | . | . | . | . | . | o | . | . | o | . | . | . | . | . | . | o | . | . | . | . | . | o | o | . | . | . | o | . | . | . | o | . |
| Guéhéneuc, 2004 | o | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | o | . | o | . | . | o | o | . | . | o | . | . | . | . | . |
| Hamou-Lhadj, 2002 | | . | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . | . | . | . | o | o | . | . | . | . | . | . | o | . | . | . | o | . |
| Hamou-Lhadj, 2003 | | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | o | . | . | o | o | . | . | . | . | . | . | o | . | . | . | o | . |
| Hamou-Lhadj, 2004 | | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | o | o | . | . | . | . | . | o | . | . | . | . | . | . |
| Hamou-Lhadj, 2005 | | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | o | . | o | o | . | . | . | . | . | . | o | . | . | . | o | . |
| Hamou-Lhadj, 2008 | | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | o | . | o | . | . | . | . | . | . | . | o | . | . | . | o | . |
| Hamou-Lhadj, 2004 | | . | . | o | . | o | . | . | . | . | o | . | . | . | o | . | . | o | . | . | o | o | . | . | . | . | . | . | . | . | . | . | . | o |
| Hamou-Lhadj, 2004 | | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o |
| Hamou-Lhadj, 2005 | | . | . | o | . | o | o | . | . | . | o | . | . | . | o | . | . | o | o | o | . | o | . | . | . | . | . | o | . | . | . | . | . | . |
| Hamou-Lhadj, 2006 | | . | . | . | . | o | . | . | . | . | o | . | . | . | o | . | . | o | o | o | . | o | . | . | . | . | . | . | o | . | . | o | o | . |

TABLE V

ARTICLE CHARACTERIZATION RESULTS (CONTINUED).

| | | activity | | | | | | | target | | | | | | method | | | | | | | | | | | | | | | | evaluation | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | tool avail. | survey | design/arch. | views | features | trace | behavior | general | legacy | procedural | oo | threads | web | distributed | vis. (std.) | vis. (adv.) | slicing | filtering | metrics | static | patt. det. | compr./summ. | heuristics | fca | querying | online | mult. traces | preliminary | regular | industrial | comparison | human subj. | quantitative | unknown/none |
| Hendrickson, 2003 | | . | . | o | . | . | o | . | . | o | . | o | . | o | o | . | . | . | . | . | . | . | o | . | o | . | . | o | . | . | . | . | . | . | . |
| Hendrickson, 2005 | | . | . | o | . | . | o | . | . | o | . | o | . | o | o | . | . | . | . | . | . | . | o | . | o | . | . | . | o | . | . | . | . | . | . |
| Heuzeroth, 2002 | | . | o | . | . | . | o | . | o | . | o | . | . | . | . | . | . | o | . | o | o | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Heuzeroth, 2003 | | . | o | . | . | . | o | . | o | . | o | . | . | . | . | . | . | o | . | o | o | . | . | . | . | . | . | . | o | . | . | . | o | . | . |
| Huang, 2005 | | . | o | . | . | . | o | . | . | . | o | . | . | . | . | . | . | o | . | o | o | . | . | . | o | . | o | . | o | . | . | . | o | . | . |
| Israr, 2007 | | . | o | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | o | o | . | . | . | . | o | . | o | . | . | . | . | . | . |
| Jerding, 1998 | | . | . | o | . | o | . | . | . | . | o | . | . | . | . | o | . | o | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Jerding, 1997 | | . | . | o | . | o | . | . | . | . | o | . | . | . | o | o | . | o | . | o | o | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Jerding, 1997 | | . | . | o | . | o | . | . | . | o | . | . | . | . | o | o | . | o | . | o | o | . | . | . | . | . | . | . | o | . | . | . | . | . | . |
| Jiang, 2007 | | . | . | . | . | . | o | . | . | o | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o |
| Jiang, 2006 | | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . |
| Jiang, 2008 | | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | o | . | . | o | o | . | . | . | . | . | o | o | o | . | o | . | . |
| Kelsen, 2004 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o |
| Kleyn, 1988 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . |
| Kollmann, 2001 | | . | . | . | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Korel, 1997 | | . | . | . | . | . | . | o | . | o | . | . | . | . | o | . | o | . | . | o | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Korel, 1998 | | . | . | . | . | . | . | o | . | o | . | . | . | . | o | o | o | . | . | o | . | o | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Koschke, 2005 | o | . | . | o | o | . | . | . | . | o | . | . | . | . | o | . | . | o | o | o | . | . | . | o | . | . | o | . | o | . | o | . | o | . | . |
| Koskimies, 1996 | o | . | . | o | . | o | . | o | . | . | o | . | . | . | o | . | . | o | . | o | o | o | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Koskinen, 2006 | | . | . | o | . | . | o | . | . | . | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Kothari, 2006 | | . | . | . | o | . | . | . | . | o | o | . | . | . | . | o | . | . | . | . | . | o | . | . | . | . | o | . | o | . | . | . | . | . | . |
| Kothari, 2007 | | . | . | . | o | . | . | . | . | o | o | . | . | . | . | . | . | . | o | . | . | o | . | . | . | . | o | . | o | . | . | . | . | . | . |
| Kuhn, 2006 | | . | . | o | o | o | . | . | . | . | o | . | . | . | . | o | . | o | o | . | . | o | . | . | . | . | o | . | o | . | . | . | . | . | . |
| Lange, 1995 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | o | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . |
| Lange, 1995 | | . | . | . | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | o | o | . | . | . | o | . | . | o | . | . | . | . | . | . | . |
| Lange, 1997 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | o | o | . | o | o | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Licata, 2003 | | . | . | . | o | . | . | . | . | . | o | . | . | . | o | . | . | . | . | o | . | o | . | . | . | . | o | . | o | . | . | . | . | . | . |
| Lienhard, 2007 | | . | . | o | o | . | . | . | . | . | o | . | . | . | . | o | . | o | . | . | . | . | . | . | . | . | . | . | o | . | o | . | o | . | . |
| Liu, 2007 | | o | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . | o | . | o | . | . | o | . | o | . | . | . | o | . | o | . | o | . | . |
| Lo, 2006 | | o | . | o | . | . | o | . | . | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o | . | o | . | o | . | . |
| Lo, 2006 | | . | . | o | . | . | o | . | . | . | o | . | . | . | o | . | . | o | o | . | o | o | o | . | . | . | . | . | o | . | o | . | o | . | . |
| Lo, 2008 | | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | o | . | . | o | . | o | . | . | . | . | . | o | . | . | . | . | . | . |
| Di Lucca, 2001 | | . | . | o | . | . | . | o | . | . | . | . | o | . | o | . | . | . | . | o | . | . | . | . | o | . | . | . | o | . | . | . | . | . | . |
| Di Lucca, 2002 | | . | . | o | . | . | . | o | . | . | . | . | o | . | o | . | . | . | . | o | . | . | . | . | o | . | . | . | o | . | . | . | . | . | . |
| Di Lucca, 2002 | | . | . | o | . | . | . | o | . | . | . | . | o | . | o | . | . | . | . | o | . | o | . | . | . | . | . | . | o | . | . | . | . | . | . |
| Di Lucca, 2003 | | . | . | o | . | . | . | o | . | . | . | . | o | . | o | . | . | . | . | o | . | o | o | . | . | . | . | . | o | . | . | . | . | . | . |
| Lukoit, 2000 | | . | . | o | o | . | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . | . | . | . | . | o | . | . | o | o | . | . | . | . |
| Martin, 2002 | | . | . | o | . | . | o | . | . | . | . | . | . | . | . | o | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o |
| Moe, 2001 | | . | . | . | . | . | . | o | . | . | . | . | . | o | . | o | . | o | o | . | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . |
| Moe, 2002 | | . | . | . | . | . | . | o | . | . | . | . | . | o | . | o | . | o | o | . | . | . | . | . | o | . | . | . | o | o | . | . | . | . | . |
| Moe, 2002 | | . | . | . | . | . | . | o | . | . | . | . | . | o | . | o | . | o | o | . | o | . | . | . | o | . | . | . | o | o | . | . | . | . | . |
| Oechsle, 2001 | | . | . | o | . | . | . | o | . | . | o | o | . | . | o | . | . | o | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Pacione, 2003 | | o | . | o | . | . | . | o | . | . | o | . | . | . | o | o | . | . | . | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . |
| Pacione, 2004 | | . | . | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| De Pauw, 1993 | | . | . | o | . | . | . | o | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o |
| De Pauw, 1994 | | . | . | o | . | . | . | o | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | o | . | o | . | . | . | . | . | . | . |
| De Pauw, 1998 | o | . | . | o | . | o | . | o | . | . | o | . | . | . | o | o | . | o | . | . | o | . | o | . | . | . | . | o | . | . | . | . | . | . | . |
| De Pauw, 2001 | o | . | . | o | . | o | . | o | . | . | o | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| De Pauw, 2006 | o | . | . | o | . | . | . | o | . | . | . | . | o | o | . | o | . | . | . | . | o | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Pheng, 2006 | | . | . | . | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | o | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . |
| Poshyvanyk, 2006 | | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . | o | o | o | . | . | o | . | . | . | o | . | o | . | o | . | . | . | . |
| Poshyvanyk, 2007 | | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . | o | o | o | . | . | o | . | . | . | o | . | o | . | o | . | o | . | . |
| Qingshan, 2005 | | . | o | o | . | . | . | . | . | . | o | . | . | . | o | . | o | . | . | o | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . |
| Quante, 2006 | | . | . | o | . | . | o | . | . | o | . | . | . | . | o | . | o | o | . | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . |
| Quante, 2007 | | . | . | o | . | . | o | . | . | . | o | o | . | . | o | . | . | o | . | . | . | . | . | . | . | . | o | . | o | . | o | . | o | . | . |

TABLE VI

ARTICLE CHARACTERIZATION RESULTS (CONTINUED).

| | | activity | | | | | | | target | | | | | | method | | | | | | | | | | | | | | | | evaluation | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | tool avail. | survey | design/arch. | views | features | trace | behavior | general | legacy | procedural | oo | threads | web | distributed | vis. (std.) | vis. (adv.) | slicing | filtering | metrics | static | patt. det. | compr./summ. | heuristics | fca | querying | online | mult. traces | preliminary | regular | industrial | comparison | human subj. | quantitative | unknown/none |
| Quante, 2008 | | . | . | o | . | . | o | o | . | o | . | . | . | . | o | . | o | o | . | . | . | . | . | . | . | . | . | . | o | . | o | . | o | . | o | . |
| Quante, 2007 | | . | . | o | . | . | o | . | . | o | o | . | . | . | o | . | . | o | o | . | o | o | . | . | . | . | . | . | o | . | o | . | o | . | . |
| Quante, 2008 | | . | . | o | o | . | . | o | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | . | . | o | . | . | o | . | . | . |
| Reiss, 2001 | | . | . | . | . | o | . | . | . | . | o | . | . | . | o | . | . | . | . | . | o | o | . | . | . | . | . | . | o | . | . | . | o | . | . |
| Reiss, 2003 | | . | . | o | . | . | . | o | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . | . |
| Reiss, 2003 | o | . | . | o | . | . | . | o | . | . | o | o | . | . | . | o | . | . | . | o | . | o | . | . | . | . | o | . | . | o | . | . | . | . | . | . |
| Reiss, 2005 | o | . | . | o | . | . | . | o | . | . | o | o | . | . | . | o | . | . | . | o | . | o | . | . | . | . | o | . | . | o | . | . | . | . | . | . |
| Reiss, 2005 | o | . | . | o | . | . | . | o | . | . | o | o | . | . | . | o | . | . | . | o | . | o | . | . | . | . | o | . | . | o | . | . | . | . | . | . |
| Reiss, 2006 | o | . | . | o | . | . | . | o | . | . | o | o | . | . | . | o | . | . | . | o | . | o | . | . | . | . | o | . | . | o | . | . | . | . | . | . |
| Reiss, 2007 | | o | . | o | . | . | . | o | . | o | o | o | . | . | o | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o |
| Renieris, 1999 | | . | . | o | . | o | . | . | . | o | . | . | . | . | . | o | . | o | . | o | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . | . |
| Richner, 1999 | | . | o | o | . | . | . | . | . | . | o | . | . | . | o | . | . | . | . | o | . | o | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Richner, 2002 | | . | . | o | . | . | . | o | . | . | o | . | . | . | . | o | . | o | . | . | o | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Rilling, 2001 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | o | o | o | o | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | o |
| Ritsch, 1993 | | . | . | . | . | . | . | o | o | . | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . | . |
| Riva, 2000 | | . | o | o | . | . | . | . | . | o | . | . | . | . | o | . | . | . | o | o | . | o | . | . | . | . | . | . | . | . | o | . | . | . | . | . |
| Riva, 2002 | | . | o | o | . | . | . | . | . | o | . | . | . | . | o | . | o | o | . | o | . | o | . | . | . | . | . | o | . | . | . | . | . | . | . | . |
| Riva, 2002 | | . | o | o | . | . | . | . | . | o | . | . | . | . | o | . | . | . | . | o | . | o | . | . | . | . | . | o | . | . | . | . | . | . | . | . |
| Rohatgi, 2008 | | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . | o | o | o | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . |
| De Roover, 2006 | | . | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Röthlisberger, 2008 | | . | . | . | . | . | . | o | . | . | o | . | . | . | . | . | . | o | . | o | . | . | . | . | . | . | . | . | o | . | . | o | . | . | . | . |
| Safyallah, 2006 | | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | . | o | o | . | . | . | . | . | o | o | . | . | . | . | . | . | . | . |
| Salah, 2003 | | . | . | o | . | . | . | o | . | . | . | . | . | o | o | . | . | . | . | o | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . | . |
| Salah, 2004 | | . | . | o | o | . | . | . | . | . | o | . | . | . | o | . | . | o | . | o | . | o | . | . | o | . | o | . | o | . | . | . | . | . | . | . |
| Salah, 2005 | | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . | . | . | o | . | o | o | . | . | o | . | o | . | o | . | . | . | . | . | . |
| Salah, 2005 | | . | . | o | o | . | . | . | . | . | o | . | . | . | o | . | . | . | o | . | . | o | . | . | . | . | o | o | . | . | . | . | . | . | . | . |
| Salah, 2006 | | . | . | o | o | . | . | . | . | . | o | . | . | . | o | . | . | . | o | . | o | o | . | . | o | . | o | . | o | . | . | . | . | . | . | . |
| Sartipi, 2006 | | . | . | . | o | . | . | . | . | o | . | . | . | . | o | . | . | o | o | o | o | o | . | o | . | . | . | . | o | . | . | . | o | . | . | . |
| Sartipi, 2006 | | . | o | o | o | . | . | . | . | o | . | . | . | . | o | . | . | . | . | o | o | o | . | o | . | . | . | . | o | . | . | . | . | . | . | . |
| Sartipi, 2007 | | . | o | o | o | . | . | . | . | o | . | . | . | . | o | . | . | . | o | o | o | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Yan, 2004 | | . | o | o | . | . | o | . | . | . | o | . | . | . | o | . | . | o | . | o | . | o | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Schmerl, 2006 | | . | o | o | . | . | o | . | . | . | o | . | . | . | o | . | . | o | . | . | o | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Sefika, 1996 | | . | o | o | . | . | . | . | . | . | o | . | . | . | o | . | . | o | . | o | . | . | . | . | o | . | . | . | o | . | . | . | . | o | . | . |
| Shevertalov, 2007 | | . | . | o | . | . | o | . | o | . | . | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | . | . | . | . | . | . | . | . | . | o |
| Simmons, 2006 | | . | . | o | o | . | . | . | . | o | . | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | o | . | o | . | . | o | o | . | . | . |
| Smit, 2008 | | . | . | o | . | . | o | . | . | . | o | . | . | . | o | . | . | o | o | . | . | o | . | . | . | . | o | . | o | . | . | . | . | . | . | . |
| Souder, 2001 | | . | . | o | . | . | . | o | . | . | o | . | . | o | o | . | . | o | . | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . | . |
| Sousa, 2007 | | . | . | . | . | . | . | . | . | . | o | . | o | . | o | . | . | o | . | o | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Stroulia, 1999 | | . | . | . | . | . | o | . | o | . | . | . | . | . | . | . | . | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | o |
| Systä, 1999 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | o | . | . | o | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Systä, 2000 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | o | . | o | o | o | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Systä, 2000 | | . | . | o | . | o | . | o | . | . | o | o | . | . | o | . | o | . | . | o | o | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Systä, 2001 | | . | . | o | . | o | . | o | . | . | o | o | . | . | o | . | o | . | o | o | o | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Tonella, 2002 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . |
| Walker, 1998 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | . | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . | . |
| Walker, 2000 | | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | . | . | o | . | . | o | . | . | . | . | . | . | . | . | . | . | o |
| Walkinshaw, 2007 | | . | . | . | . | . | o | . | . | . | o | . | . | . | o | . | . | . | . | . | o | . | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Walkinshaw, 2008 | | . | . | . | . | . | o | . | . | . | o | . | . | . | o | . | . | . | o | o | o | o | . | . | o | . | . | . | o | . | . | . | . | . | . | . |
| Wang, 2005 | | . | o | . | . | . | . | . | . | . | o | . | . | . | . | . | . | . | . | o | o | . | . | . | o | . | . | . | o | . | . | . | . | o | . | . |
| Wilde, 1996 | o | . | . | . | o | . | . | o | . | o | . | . | . | . | . | . | . | . | o | . | . | . | o | . | . | . | o | . | o | . | o | . | . | . | . | . |
| Wilde, 1999 | | . | . | . | . | . | . | o | . | o | . | . | . | . | . | . | . | . | o | . | . | . | o | . | . | . | o | o | . | . | . | . | . | . | . | . |
| Wilde, 2001 | o | . | . | . | o | . | . | . | o | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o | o | o | . | . | . | . |
| Wilde, 2003 | o | . | . | . | o | . | . | . | o | o | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | o | o | o | . | . | . | . |
| Wilde, 1995 | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | . | . | o | . | . | . | o | . | . | . | o | o | . | . | . | o | . | . | . | . |
| Wong, 1999 | o | . | . | . | o | . | . | . | . | o | . | . | . | . | o | . | o | . | . | o | . | . | o | . | . | . | . | . | o | . | . | . | . | o | . | . |

TABLE VII

ARTICLE CHARACTERIZATION RESULTS (CONTINUED).

| | | activity | | | | | | | target | | | | | | method | | | | | | | | | | | | | | | evaluation | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | tool avail. | survey | design/arch. | views | features | trace | behavior | general | legacy | procedural | oo | threads | web | distributed | vis. (std.) | vis. (adv.) | slicing | filtering | metrics | static | patt. det. | compr./summ. | heuristics | fca | querying | online | mult. traces | preliminary | regular | industrial | comparison | human subj. | quantitative | unknown/none |
| Wong, 2000 | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | o | . | o | o | . | . | o | . | . | . | . | . | o | . | . | . | . | o | . |
| Wong, 2005 | o | . | . | . | o | . | . | . | . | o | . | . | . | . | . | . | o | . | o | o | . | . | o | . | . | . | . | . | o | . | . | . | . | o | . |
| Zaidman, 2004 | | . | . | . | . | o | . | . | . | . | o | . | . | . | . | o | . | o | o | . | . | o | o | . | . | . | . | . | o | . | . | . | . | . | . |
| Zaidman, 2005 | | . | . | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . | o | . | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . |
| Zaidman, 2006 | | . | . | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . | o | o | . | . | o | . | . | . | . | . | o | . | . | . | . | . | . |
| Zaidman, 2008 | | . | . | . | . | . | . | o | . | . | o | . | . | . | . | . | . | . | o | o | . | . | o | . | . | . | . | . | o | . | . | . | . | o | . |
| Zaidman, 2006 | | . | . | . | . | o | . | o | o | o | . | . | . | . | . | o | . | . | o | . | . | . | o | . | . | . | . | . | . | . | o | . | . | . | . |

TUD-SERG-2008-033
ISSN 1872-5392