A Systematic Survey of Program Comprehension through Dynamic Analysis

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

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 176 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—Survey, program comprehension, dynamic analysis.

1 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 percent of the software engineering effort is spent on understanding the software system at hand [1], [2].

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 [3], [4], [5]. 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.

- *B. Cornelissen, A. Zaidman, and A. van Deursen are with the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628CD Delft, The Netherlands. E-mail: {s.g.m.cornelissen, a.e.zaidman, arie.vandeursen}@tudelft.nl.*
- *L. Moonen is with Simula Research Laboratory, PO Box 134, 1235 Lysaker, Norway. E-mail: leon.moonen@computer.org.*
- *R. Koschke is with Arbeitsgruppe Softwaretechnik, Universität Bremen, Postfach 33 04 40, 28334 Bremen, Germany. E-mail: koschke@informatik.uni-bremen.de.*

Manuscript received 22 Oct. 2008; accepted 24 Mar. 2009; published online 3 Apr. 2009.
Recommended for acceptance by H. Muller.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2008-10-0350.
Digital Object Identifier no. 10.1109/TSE.2009.28.

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 are 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).

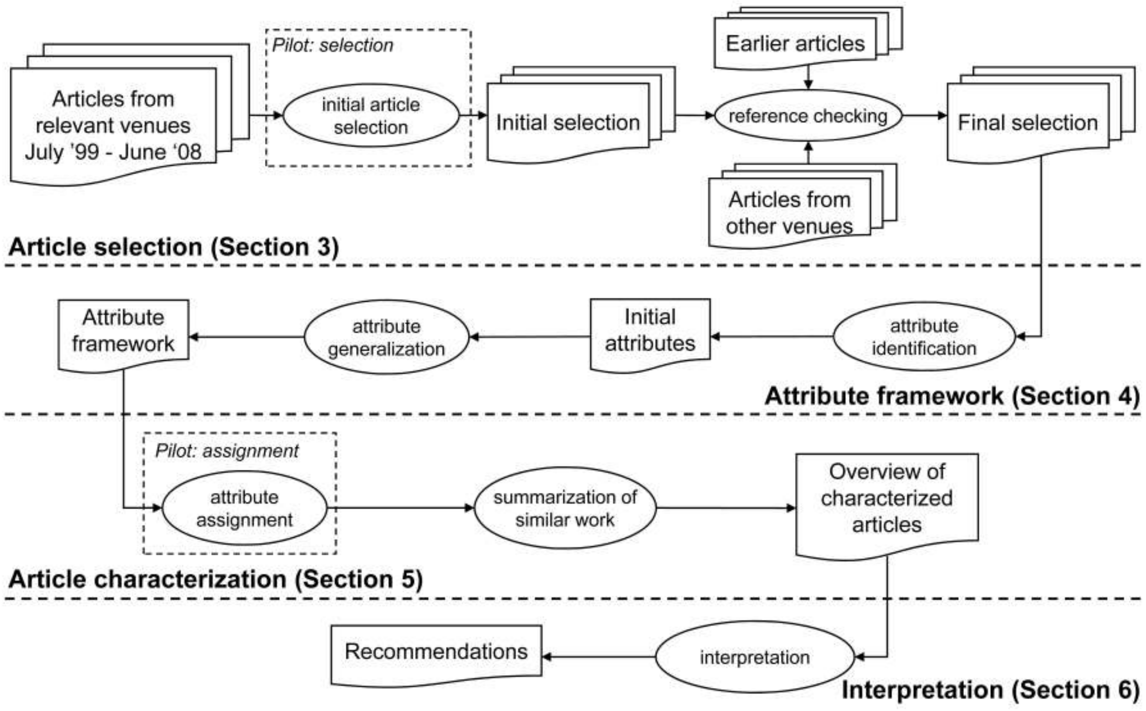
Articles from relevant venues July '99 - June '08
Pilot: selection
initial article selection
Initial selection
Earlier articles
reference checking
Articles from other venues
Final selection
Article selection (Section 3)
Attribute framework
attribute generalization
Initial attributes
attribute identification
Attribute framework (Section 4)
Pilot: assignment
attribute assignment
summarization of similar work
Overview of characterized articles
Article characterization (Section 5)
Recommendations
interpretation
Interpretation (Section 6)

Fig. 1. Overview of the systematic survey process.

Within each facet, one can distinguish a series of generic *attributes*: The examples given above (in parentheses) 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 2 presents an introduction on dynamic analysis for program comprehension. The protocol that lies at the basis of our survey is outlined in Fig. 1, which distinguishes four phases that are described in Sections 3-6. Section 7 evaluates our approach and findings and, in Section 8, we conclude with a summary of the key contributions of this paper.

2 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.

2.1 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"* [6]. 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"* [7]. Note that this definition remains purposely 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 on the benefits and limitations of using dynamic analysis in program comprehension contexts. The benefits that we consider are:

- The *precision* with regard to the actual behavior of the software system, for example, in the context of object-oriented 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 behaviors or traces under analysis capture 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 cases where timing issues play a role. Examples include multithreaded and multiprocess programs [8], real-time software, and device drivers.

In order to deal with these limitations, many techniques propose abstractions or heuristics, allowing grouping of 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?).

2.2 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 profiling is used to measure (and optimize) the 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 [9]. 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 [10] 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. [11], [12], [13] started their extensive (and still ongoing) 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 [14] 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 [15] 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 [16], [17], [18]. At the same time, Lange and Nakamura [19], [20] 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 [21] 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. [22] reasoned from a higher level of abstraction in their efforts to generate architecture-oriented visualizations.

In 1997, Jerding et al. [23], [24] proposed their well-known ISVis tool to visualize large execution traces. Two linked views were offered: a continuous sequence diagram and the "information mural" [25]: a dense, navigable representation of an entire trace.

Walker et al. [5] 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 [26]).

Finally, in 1999, Ball [7] 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 [3] used static and dynamic information to reconstruct architectural views. They continued this work later on [27], with their focus shifting to the recovery of collaboration diagrams with Prolog queries in their Collaboration Browser tool.

1. The Eclipse Test and Performance Tools Platform Project, http://www.eclipse.org/tptp/.

2.3 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. [28], Systä et al. [29], and Briand et al. [30]. 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 [31] and Hamou-Lhadj et al. [32], [33], [34].

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. [35], [36], who combine static and dynamic analyzes to detect design patterns in legacy code. Also of interest is the work on architecture reconstruction by Riva et al. [37], [38], and DiscoTect, a tool by Schmerl et al. [4], [39] 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. [40], who use behavioral profiles to illustrate architecturally significant behavioral rules, and an article by Cook and Du [41] in which thread interactions are exposed in distributed systems. Furthermore, recently, there has been considerable effort in the recovery of protocols [42], specifications [43], and grammars [44].

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. [45], [46] and Kothari et al. [47], particularly the activity of feature location has become increasingly popular since the aforementioned work by Wilde and Scully [15]. Influential examples include techniques by Wong et al. [48] (using execution slices), Eisenbarth et al. [49] (using formal concept analysis), Antoniol and Guéhéneuc [50] (through statistical analyzes), and Poshyvanyk et al. [51] (using complementary techniques).

2.4 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 identify future directions, there is a 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 [32] discussed eight trace exploration tools in terms of three criteria: trace modeling, abstraction level, and size reduction. In the same year, Pacione et al. [52] evaluated five dynamic visualization tools on a series of program comprehension tasks. Greevy's PhD thesis [53] from 2007 summarized several directions within program comprehension, with an emphasis on feature analysis. Also from 2007 is a study by Reiss [54], 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. Fig. 1 shows the tasks involved, which are discussed in the following sections.

3 Article Selection

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

3.1 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.

3.1.1 Identification of Research

Search strategies in literature surveys often involve automatic keyword searches (e.g., [55], [56]). However, Brereton et al. [57] 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 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. [58], 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 1 to be the most closely related to

TABLE 1
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

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 in any of the relevant venues as a full paper or a short paper.

It could be argued that the International Workshop on Program Comprehension through Dynamic Analysis and the International Workshop on Dynamic Analysis should also be included for their relevance to the topic. However, as most of the (influential) papers from these two workshops were republished later on, we will only consider journals and conferences in this survey.

3.1.2 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 [57]; borderline cases are resolved by discussion among the authors.

3.2 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 [59] and Brereton et al. [57], 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 preselected *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]

The domain experts who serve as raters in the pilot are the last three authors of this survey. Since these authors 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.

3.2.1 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.

3.3 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 (nonrecursively) extend the initial selection with relevant articles that have been cited therein, regardless of publication date and venue but taking the

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

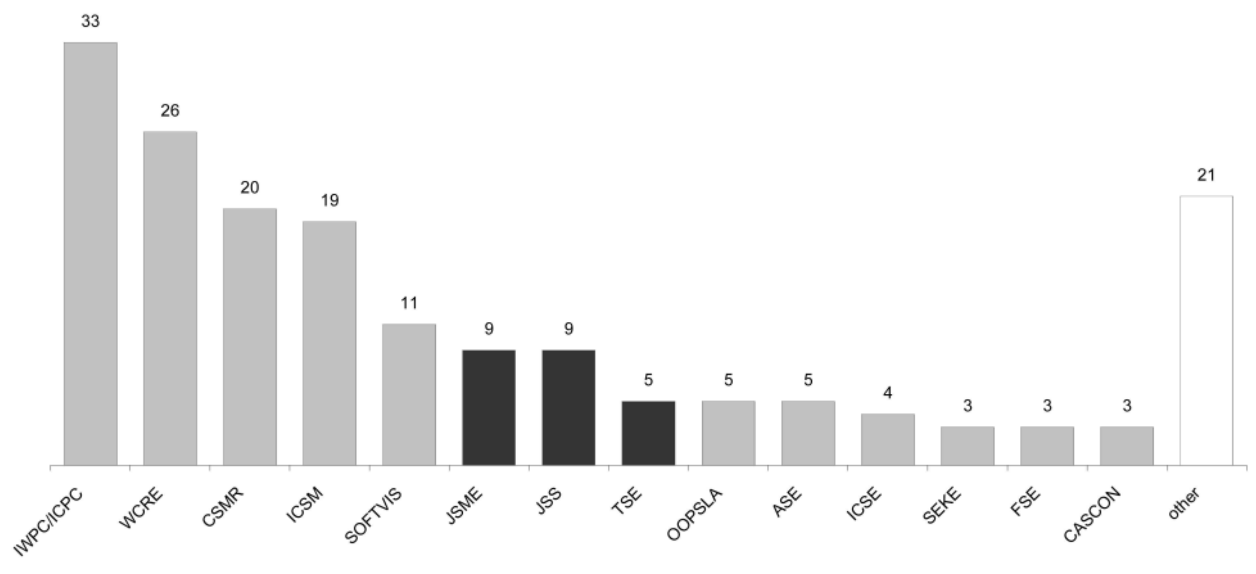

Fig. 2. Distribution of the final article selection across the different venues. Dark bars denote journals; light bars denote conferences.

selection criteria into account. This procedure minimizes the chance of influential literature being missed and results in a *final* article selection.

3.4 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 1. The reference checking yielded another 49 articles (and 17 additional venues), which were subsequently included in the selection. Interestingly, we also identified quite a number of papers from the ACM Symposium on Software Visualization (SOFTVIS); to avoid missing too many papers from this venue, we decided to include it in our systematic process.

The end result is a research body that is comprised of 176 articles. The full listing of these articles is available online[3] and in a technical report [60]. Fig. 2 shows the distribution of all surveyed articles across the venues from which at least three articles were selected.

4 Attribute Framework

As shown in Fig. 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.

4.1 Attribute Identification

As stated in Section 1, 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.

4.2 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.

4.3 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 2 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.

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

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

TABLE 2
Attribute Framework

Facet	Attribute	Description
Activity	survey	a survey or comparative evaluation of existing soltuions fulfilling 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.
	case study	case studies 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.

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 as such a classification greatly depends 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 [25]). 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 "case study" 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).

5 Article Characterization

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

5.1 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 attempt to 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. As we discuss later on, legacy systems are an exception because their definition is rather vague.

5.2 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 with the loss of virtually no information. The listing and characterization of the discarded articles are available in the aforementioned technical report and Web site.

5.3 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 is 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 2.

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.

5.4 Characterization Pilot Results and 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 2.

As an example of some of the adjustments within the activity facet, we renamed our original "communication" attribute to "behavior" and decided to remove our original "framework" attribute because determining whether an article constitutes a framework often turned out to be difficult. Within the method facet, we had some discussions on trace comparisons and decided to call the attribute "multiple traces" to make it more general. Furthermore, we unified compression, merging, and clustering techniques into "compression/summarization" and included selective tracing into the "filtering" attribute, as the distinction between the two is generally subtle and largely dependent on the manner in which these techniques are described by the authors. Within the evaluation facet, we decided to add "open source" to the description of "case study" and explicitly mention the use of questionnaires as an evaluation approach.

A full listing and description of the changes made is given in the technical report [60].

5.5 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.

5.6 Characterization Results

The characterization and summarization of the 176 selected articles resulted in an overview of 114 articles, as shown in Tables 3 and 4. 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

TABLE 3
Article Characterization Results

			activity							target						method													evaluation						
	add'l articles	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	case study	industrial	comparison	human subj.	quantitative	unknown/none
Antoniol, 2004 [84]			.	.	.	.	o	.	.	.	o	o	.	.	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.	.	o	.
Antoniol, 2004 [85]			.	o	o	.	.	.	.	.	.	.	.	o	.	o	.	.	o	.	o	.	.	.	.	.	.	.	.	o	.	.	.	o	.
Antoniol, 2006 [50]	1		o	.	.	o	.	.	.	.	o	o	o	.	.	.	.	.	o	.	o	.	.	.	.	.	.	o	.	o	.	o	.	o	.
Ball, 1996 [86]			.	.	o	.	.	.	o	.	o	.	.	.	.	.	o	o	.	o	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.
Ball, 1999 [7]			.	.	.	.	.	.	o	.	o	.	.	.	.	.	.	.	.	o	o	.	.	.	o	.	.	.	o	.	.	.	.	.	.
Bennett, 2008 [77]			o	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	o	.	.	o	o	.	.	.	.	.	.	o	.	.	o	.	.
Bohnet, 2006 [87]			.	.	.	o	.	.	.	.	o	o	.	.	.	.	o	o	.	.	o	o	.	.	.	.	.	.	o	.	.	.	.	.	.
Bojic, 2000 [88]			.	o	o	.	.	.	.	.	.	o	.	.	.	o	.	.	.	.	.	.	o	.	o	.	.	.	o	.	.	.	.	.	.
Briand, 2006 [30]	2		o	.	o	.	.	.	o	.	.	o	o	.	o	o	.	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.	.	o	.
Chan, 2003 [89]		o	.	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	.	.	.	.	.
Cook, 2005 [41]	2		.	.	.	.	.	o	.	.	o	.	o	.	.	o	.	.	.	o	.	o	.	o	.	.	.	.	.	o	.	.	.	.	.
Cornelissen, 2007 [90]			.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	o	.	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.
Cornelissen, 2008 [91]	1	o	.	.	o	o	o	.	o	.	.	o	.	.	.	o	o	.	o	.	o	.	o	.	.	.	.	.	.	o	o	.	.	.	.
Cornelissen, 2008 [81]			o	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	o	o	.	o	o	o	.	.	.	o	.	o	.	.	.	o	.
Dalton, 2008 [92]			.	.	o	.	.	.	o	.	.	.	.	.	.	o	.	.	o	.	o	.	.	.	.	.	.	.	o	.	.	.	.	o	.
Deprez, 2000 [93]			.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	.	o	o	.	.	.	.	.	.
Ducasse, 2004 [94]		o	.	.	o	.	.	.	o	.	.	o	.	.	.	.	o	.	o	o	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.
Eaddy, 2008 [80]		o	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	o	o	.	o	.	.	o	.	o	.	o	.	o	.	o	.	o	.
Edwards, 2006 [95]			.	.	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	o	.	o	o	.	.	.	.
Eisenbarth, 2003 [49]	4		.	.	o	o	.	.	.	.	o	.	.	.	.	o	.	.	o	.	o	.	.	.	o	.	.	o	.	o	o	.	.	.	.
Eisenberg, 2005 [96]			.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.	.	o	o	.	.	o	.	o	.	.	.	o	.	o	.	o	.
El-Ramly, 2002 [97]	3		.	.	.	o	.	o	.	o	.	.	.	.	.	o	.	.	.	o	.	o	o	.	.	.	.	.	.	.	o	o	.	o	.
Fischer, 2005 [98]			.	.	o	.	o	.	.	.	o	.	o	.	.	.	o	.	.	o	.	.	.	.	.	.	.	o	.	o	.	.	.	.	.
Fisher II, 2007 [99]			.	.	.	.	.	o	.	.	.	.	.	o	.	.	.	.	o	.	.	.	.	o	.	.	.	.	.	o	.	.	.	o	.
Gargiulo, 2001 [100]		o	.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.
Gestwicki, 2005 [101]			.	.	o	.	.	.	o	.	.	o	o	.	.	o	.	.	.	.	.	.	.	.	.	o	.	o	.	.	.	.	.	.	o
Greevy, 2006 [46]	1		.	.	o	o	.	.	.	.	.	o	.	.	.	.	o	.	o	o	.	.	o	.	.	.	.	o	.	o	.	.	.	.	.
Greevy, 2005 [45]			.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.	o	o	o	.	.	.	.	.	.	o	.	o	.	.	.	.	.
Greevy, 2006 [102]	1		.	.	o	o	.	.	.	.	.	o	.	.	.	.	o	.	o	o	o	.	.	.	.	o	.	o	.	o	.	.	.	.	.
Gschwind, 2003 [103]	1		.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	o	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.
Guéhéneuc, 2002 [104]		o	.	.	.	.	.	.	o	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	.	o	o	.	.	o	.	.	.	o	.
Guéhéneuc, 2004 [105]		o	.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	o	.	o	.	o	.	.	o	o	.	.	o	.	.	.	.	.
Hamou-Lhadj, 2004 [34]	5		.	.	o	.	o	.	.	.	o	o	.	.	.	o	.	.	o	o	.	o	o	.	.	.	.	.	.	o	.	.	.	o	o
Hamou-Lhadj, 2004 [32]			o	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o
Hamou-Lhadj, 2006 [33]	1		.	.	o	.	o	o	.	.	.	o	.	.	.	o	.	.	o	o	o	.	o	.	.	.	.	.	.	o	.	.	o	o	.
Hendrickson, 2005 [106]	1		.	.	o	.	.	o	.	.	o	.	o	.	o	o	.	.	.	.	.	.	.	o	.	o	.	.	.	o	.	.	.	.	.
Heuzeroth, 2003 [36]	1		.	o	.	.	.	o	.	o	.	o	.	.	.	.	.	.	o	.	o	o	.	.	.	.	.	.	.	o	.	.	.	o	.
Huang, 2005 [107]			.	o	.	.	.	o	.	.	.	o	.	.	.	.	.	.	o	.	o	o	.	.	.	o	.	o	.	o	.	.	.	o	.
Israr, 2007 [64]			.	o	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	.	o	o	.	.	.	.	o	.	o	.	.	.	.	.
Jerding, 1997 [23]	2		.	.	o	.	o	.	.	.	o	o	.	.	.	o	o	.	o	.	o	o	.	.	.	.	.	.	.	o	.	.	.	.	.
Jiang, 2007 [108]			.	.	.	.	.	o	.	.	o	.	.	.	.	o	.	.	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o
Jiang, 2006 [109]			.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.
Jiang, 2008 [110]			.	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.	o	o	.	.	.	.	.	o	o	o	.	o	.
Kelsen, 2004 [111]			.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	.	.	.	o
Kleyn, 1988 [10]			.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	.
Kollmann, 2001 [112]			.	.	.	.	.	.	o	.	.	o	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.
Korel, 1998 [113]	1		.	.	.	.	.	.	o	.	o	.	.	.	.	o	o	o	.	.	o	.	o	.	.	.	.	.	o	.	.	.	.	.	.
Koschke, 2005 [114]		o	.	.	o	o	.	.	.	.	o	.	.	.	.	o	.	.	o	o	o	.	.	.	o	.	.	o	.	o	.	o	.	o	.
Koskimies, 1996 [21]		o	.	.	o	.	o	.	o	.	.	o	.	.	.	o	.	.	o	.	o	o	o	.	.	.	.	.	o	.	.	.	.	.	.
Koskinen, 2006 [40]			.	.	o	.	.	o	.	.	.	o	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.
Kothari, 2007 [47]	1		.	.	.	o	.	.	.	.	o	o	.	.	.	.	o	.	.	o	.	.	o	.	.	.	.	o	.	o	.	.	.	.	.
Kuhn, 2006 [115]			.	.	o	o	o	.	.	.	.	o	.	.	.	.	o	.	o	o	.	.	o	.	.	.	.	o	.	o	.	.	.	.	.
Lange, 1995 [19]	1		.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	o	.	o	o	.	.	.	o	.	.	o	.	.	.	.	.	.
Lange, 1997 [116]			.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	o	o	.	o	o	.	.	.	.	.	.	o	.	.	.	.	.	.
Licata, 2003 [117]			.	.	.	o	.	.	.	.	.	o	.	.	.	o	.	.	.	.	o	.	o	.	.	.	.	o	.	o	.	.	.	.	.
Lienhard, 2007 [118]			.	.	o	o	.	.	.	.	.	o	.	.	.	.	o	.	o	.	.	.	.	.	.	.	.	.	.	o	.	o	.	o	.
Liu, 2007 [61]			o	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.	o	.	o	.	.	o	.	o	.	.	.	o	.	o	.	o	.

value for the facet at hand can be considered "other," "unknown," or "none." The characterization of *all* 176 articles is available online and in the aforementioned technical report [60]; in the remainder of this survey, however, we speak only in terms of the 114 *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

TABLE 4
Article Characterization Results (Continued)

			activity							target						method													evaluation						
	add'l articles	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	case study	industrial	comparison	human subj.	quantitative	unknown/none
Lo, 2006 [119]			o	.	o	.	.	o	.	.	.	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	o	.
Lo, 2006 [120]			.	.	o	.	.	o	.	.	.	o	.	.	.	o	.	.	o	o	.	o	o	o	.	.	.	.	.	o	.	o	.	o	.
Lo, 2008 [43]			.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	o	.	.	o	.	o	.	.	.	.	.	o	.	.	.	.	.
Di Lucca, 2003 [121]	3		.	.	o	.	.	.	o	.	.	.	.	o	.	o	.	.	.	.	o	.	o	o	.	o	.	.	o	o	.	.	.	.	.
Lukoit, 2000 [122]			.	.	o	o	.	.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.	.	.	.	o	.	.	o	o	.	.	.	.
Malloy, 2005 [123]			.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	o	.	o	.	.	.	.	.	o	.	o	.	.	.	.	.	.
Martin, 2002 [124]			.	.	o	.	.	o	.	.	.	.	.	.	.	.	o	.	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o
Moe, 2002 [125]	2		.	.	.	.	.	.	o	.	.	.	.	.	o	.	o	.	o	o	.	o	.	.	.	o	.	.	.	o	o	.	.	.	.
Oechsle, 2001 [126]			.	.	o	.	.	.	o	.	.	o	o	.	.	o	.	.	o	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.
Orso, 2003 [127]			.	.	o	.	.	.	o	.	.	o	.	.	.	.	o	.	o	.	o	.	o	.	.	.	.	.	o	o	.	.	.	.	.
Pacione, 2003 [52]			o	.	o	.	.	.	o	.	.	o	.	.	.	o	o	.	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.
Pacione, 2004 [128]			.	.	.	.	.	.	o	.	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.
De Pauw, 1994 [12]	1		.	.	o	.	.	.	o	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	o	.	.	o	.	.	.	.	.	.
De Pauw, 2001 [28]	1	o	.	.	o	.	o	.	o	.	.	o	o	.	.	o	o	.	o	.	.	o	.	o	.	.	.	.	o	.	.	.	.	.	.
De Pauw, 2006 [129]		o	.	.	o	.	.	.	o	.	.	.	.	o	o	.	o	.	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.	.	.
Pheng, 2006 [130]			.	.	.	.	.	.	o	.	.	o	.	.	.	o	.	.	.	.	o	.	.	o	.	.	.	.	.	o	.	.	.	.	.
Poshyvanyk, 2007 [51]	1		.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.	o	o	o	.	.	o	.	.	.	o	.	o	.	o	.	o	.
Qingshan, 2005 [131]			.	o	o	.	.	.	.	.	.	o	.	.	.	o	.	o	.	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	.
Quante, 2007 [42]	3		.	.	o	.	.	o	o	.	o	o	o	.	.	o	.	o	o	o	.	o	o	.	.	.	o	.	.	o	.	o	.	o	.
Quante, 2008 [78]			.	.	o	o	.	.	o	.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	.	.	.	.	o	.	.	o	.	.
Reiss, 2001 [31]			.	.	.	.	o	.	.	.	.	o	.	.	.	o	.	.	.	.	.	o	o	.	.	.	.	.	.	o	.	.	.	o	.
Reiss, 2003 [132]			.	.	o	.	.	.	o	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.
Reiss, 2006 [133]	3	o	.	.	o	.	.	.	o	.	.	o	o	.	.	.	o	.	.	.	o	.	o	.	.	.	o	.	.	o	.	.	.	.	.
Reiss, 2007 [54]			o	.	o	.	.	.	o	.	o	o	o	.	.	o	o	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	o
Renieris, 1999 [134]			.	.	o	.	o	.	.	.	o	.	.	.	.	.	o	.	o	.	o	.	.	.	.	o	.	.	o	.	.	.	.	.	.
Richner, 1999 [3]			.	o	o	.	.	.	.	.	.	o	.	.	.	o	.	.	.	.	o	.	o	.	.	o	.	.	.	o	.	.	.	.	.
Richner, 2002 [27]			.	.	o	.	.	.	o	.	.	o	.	.	.	.	o	.	o	.	.	o	.	.	.	o	.	.	.	o	.	.	.	.	.
Rilling, 2001 [135]			.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	o	o	o	o	.	o	.	.	.	.	.	.	.	.	.	.	.	o
Ritsch, 1993 [136]			.	.	.	.	.	.	o	o	.	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.
Riva, 2000 [137]			.	o	o	.	.	.	.	.	o	.	.	.	.	o	.	.	.	o	o	.	o	.	.	.	.	.	.	.	o	.	.	.	.
Riva, 2002 [38]	1		.	o	o	.	.	.	.	.	o	.	.	.	.	o	.	o	o	.	o	.	o	.	.	.	.	.	o	.	.	.	.	.	.
Rohatgi, 2008 [138]			.	.	.	o	.	.	.	.	.	o	.	.	.	.	.	.	o	o	o	.	.	o	.	.	.	.	o	.	.	.	.	.	.
De Roover, 2006 [139]			.	.	.	.	.	o	.	.	o	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	.
Röthlisberger, 2008 [72]			.	.	.	.	.	.	o	.	.	o	.	.	.	.	.	.	o	.	o	.	.	.	.	.	.	.	.	o	.	.	o	.	.
Safyallah, 2006 [140]			.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	.	o	o	.	.	.	.	.	o	o	.	.	.	.	.	.
Salah, 2003 [141]			.	.	o	.	.	.	o	.	.	.	.	.	o	o	.	.	.	.	o	.	.	.	.	o	.	.	o	.	.	.	.	.	.
Salah, 2004 [142]			.	.	o	o	.	.	.	.	.	o	.	.	.	o	.	.	o	.	o	.	o	.	.	o	.	o	.	o	.	.	.	.	.
Salah, 2006 [143]	2		.	.	o	o	.	.	.	.	.	o	.	.	.	o	.	.	.	o	.	o	o	.	.	o	.	o	.	o	.	.	.	.	.
Sartipi, 2007 [144]	2		.	o	o	o	.	.	.	.	o	.	.	.	.	o	.	.	o	o	o	o	o	.	o	o	.	.	.	o	.	.	.	o	.
Schmerl, 2006 [4]	1		.	o	o	.	.	o	.	.	.	o	.	.	.	o	.	.	o	.	.	o	.	.	.	o	.	.	.	o	.	.	.	.	.
Sefika, 1996 [22]			.	o	o	.	.	.	.	.	.	o	.	.	.	o	.	.	o	.	o	.	.	.	.	o	.	.	.	o	.	.	.	o	.
Shevertalov, 2007 [145]			.	.	o	.	.	o	.	o	.	.	.	.	.	o	.	.	o	.	.	.	o	.	.	.	.	.	.	.	.	.	.	.	o
Simmons, 2006 [79]			.	.	o	o	.	.	.	.	o	.	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	o	.	o	.	.	o	o	.
Smit, 2008 [146]			.	.	o	.	.	o	.	.	.	o	.	.	.	o	.	.	o	o	.	.	o	.	.	.	.	o	.	o	.	.	.	.	.
Souder, 2001 [147]			.	.	o	.	.	.	o	.	.	o	.	.	o	o	.	.	o	.	.	.	.	.	.	.	.	.	o	.	.	.	.	.	.
Sousa, 2007 [148]			.	.	.	.	.	.	.	.	.	o	.	o	.	o	.	.	o	.	.	o	.	.	.	.	.	.	.	o	.	.	.	.	.
Stroulia, 1999 [149]			.	.	.	.	.	o	.	o	.	.	.	.	.	.	.	.	o	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	o
Systä, 2001 [29]	3		.	.	o	.	o	.	o	.	.	o	o	.	.	o	.	o	.	o	o	o	.	.	.	o	.	.	.	o	.	.	.	.	.
Tonella, 2002 [150]			.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	.	.	o	.	.	.	.	.	.	.	.	o	.	.	.	.	.
Walker, 2000 [151]	1		.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	o	.	.	.	o	.	.	o	.	.	o	.	.	.	.	.	o
Walkinshaw, 2008 [44]	1		.	.	.	.	.	o	.	.	.	o	.	.	.	o	.	.	.	o	o	o	o	.	.	o	.	.	.	o	.	.	.	.	.
Wang, 2005 [152]			.	o	.	.	.	.	.	.	.	o	.	.	.	.	.	.	.	.	o	o	.	.	.	o	.	.	.	o	.	.	.	o	.
Wilde, 1995 [15]	4	o	.	.	.	o	.	.	o	o	o	.	.	.	.	.	.	.	.	o	.	.	.	o	.	.	.	o	o	o	o	o	o	.	.
Wong, 2005 [153]	2	o	.	.	.	o	.	.	.	.	o	.	.	.	.	o	.	o	.	o	o	.	.	o	.	.	.	.	.	o	.	.	.	o	.
Zaidman, 2004 [62]			.	.	.	.	o	.	.	.	.	o	.	.	.	.	o	.	o	o	.	.	o	o	.	.	.	.	.	o	.	.	.	.	.
Zaidman, 2006 [70]			.	.	.	.	o	.	o	o	o	.	.	.	.	.	o	.	.	o	.	.	.	o	.	.	.	.	.	.	o	.	.	.	.
Zaidman, 2008 [154]	2		.	.	.	.	.	.	o	.	.	o	.	.	.	.	.	.	.	o	o	.	.	o	.	.	.	.	.	o	.	.	.	o	.

work section is of such quality that it constitutes a respectable survey (e.g., [30], [61]).

The overview in Tables 3 and 4 serves as a useful reference when seeking related work in particular subfields. For example, when looking for literature on trace visualization, one needs only to 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.

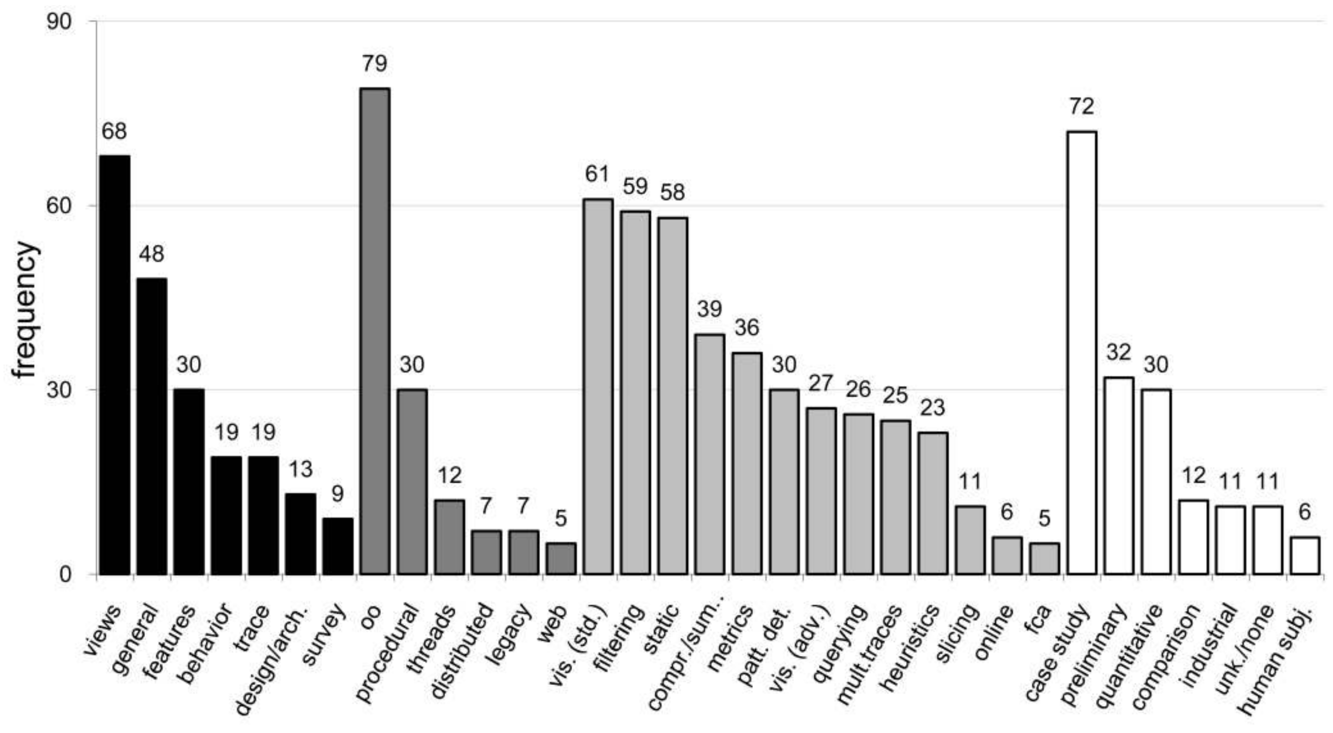

Fig. 3. Distribution of the attributes in each facet across the summarized articles. From left to right: activity, target, method, and evaluation.

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. The full results of this experiment are given in the technical report.

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

6 Avenues for Future Research

Given the article selection and attribute assignments of Tables 3 and 4, our final survey step (see Fig. 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 interesting attribute combinations. In this section, we offer a selection of the most important outcomes of this analysis.

6.1 Most Common Attributes

Understanding the most common attributes (as displayed in Fig. 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 [62]. 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: 79 out of the 114 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 be bacuse of 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 analyzes are combined [63].

Finally, within the evaluation facet, we note that case studies (typically, using open-source systems) are the most typical and that comparisons, industrial 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 two and half years. Interestingly, more than half of these evaluations involved the *feature location* activity; this is further discussed in Section 6.5.

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.

6.2 Least Common Activities

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

As discussed in Section 2.4, 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 5.6).

In our survey, there 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 tracing techniques, e.g., registering method calls, from which occurrences of design patterns such as the Observer or Mediator can be identified [36].

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 [4], [22], [64]. 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.

6.3 Least Common Targets

6.3.1 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 Web server 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 fill them in automatically based on a small number of default values provided by the software engineer [65].

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. [66]. 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.

6.3.2 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" [67]. Furthermore, distributed systems often behave differently than intended, because of unanticipated usage patterns that are a direct consequence of their dynamic configurability [68]. This increases the need to understand these systems, and due to their heterogeneous nature, dynamic analysis constitutes a viable approach.

6.3.3 Multithreaded Applications

In recent years, multicore CPUs have become mainstream hardware and multithreading has become increasingly important. The evolution toward 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. [69] 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 12 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: Ddoes 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.

6.3.4 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 [70].

6.4 Least Common Evaluations

6.4.1 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 [71].

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. [72] who provide dynamic information during programming tasks.

Another impediment for industrial involvement in publications can be fear of 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., [73]) or capture/replay techniques (e.g., [74], [75]).

6.4.2 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 [76].

In spite of its importance, this type of evaluation was used in no more than six articles. Bennett et al. [77] use four experts and five graduate students to assess the usefulness of reverse engineered UML sequence diagrams in nine specific comprehension tasks. Quante [78] reports on a controlled experiment with 25 students that involves the use of "object process graphs" in a program comprehension context. Röthlisberger et al. [72] 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 [33] 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. [16] and Simmons et al. [79] conduct experiments to assess the practical usefulness of different feature location techniques in legacy Fortran software and 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.

6.4.3 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. [80] 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 [50] and Poshyvanyk et al. [51]; in the same context, Wilde et al. [16] 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 114 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 [81], 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. [82] 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., [81]), and common evaluation criteria, such as the precision and recall measures that are often used in the field of feature location (e.g., [80]). Also of importance in this respect is the use of open-source cases to enable the reproducibility of experiments.

6.5 How Activities Are Evaluated

In the historical overview in Section 2, 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 6.4. The introduction of common evaluation criteria (i.e., precision and recall) may have contributed to this development. Furthermore, feature analysis accounts for 7 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., [11], [21]) and has only recently been subjected to a controlled experiment [77]. 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 [81].

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 is comprised of 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 the 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 [42], [43].

7 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 [59] and Brereton et al. [57]. 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.

7.1 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 of 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 [59] and Brereton et al. [57], 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 3.2).

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. [57], who state that "current software engineering search engines are not designed to support systematic literature reviews;" this observation is confirmed by Staples and Niazi [83]. 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 1 were chosen because they are most closely related to software engineering, maintenance, and reverse engineering. While this presumption is largely supported by the results (Fig. 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 nonrecursive 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.

7.2 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.

7.3 Article Characterization

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

We have addressed this validity threat through a second pilot study (Section 5.3), 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; a detailed description of these refinements is given in the technical report [60].

In order to identify topics that have received little attention in the literature, we counted the occurrences of all attributes in the selected articles, as shown in Fig. 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, while making the full selection available in a technical report and on a Web site.

7.4 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 toward the analysis of Tables 3 and 4: 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.

8 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 is comprised of 31 venues and relevant articles of up to 30 years old.

Out of 4,795 scanned papers 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 176 papers. 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 to 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;
4. the distillation of future directions.

The identified attributes are shown in Table 2. 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 3 and 4. The frequencies of the attributes are provided by Fig. 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 significant technical progress in the past decade (Section 6.5). 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 (Sections 6.1 and 6.3). 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 6.4).

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.

ACKNOWLEDGMENTS

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

REFERENCES

[1] T.A. Corbi, "Program Understanding: Challenge for the 1990s," *IBM Systems J.*, vol. 28, no. 2, pp. 294-306, 1989.

[2] R.K. Fjeldstad and W.T. Hamlen, "Application Program Maintenance Study: Report to Our Respondents," *Proc. GUIDE*, vol. 48, 1979.

[3] T. Richner and S. Ducasse, "Recovering High-Level Views of Object-Oriented Applications from Static and Dynamic Information," *Proc. 15th Int'l Conf. Software Maintenance*, pp. 13-22, 1999.

[4] B.R. Schmerl, J. Aldrich, D. Garlan, R. Kazman, and H. Yan, "Discovering Architectures from Running Systems," *IEEE Trans. Software Eng.*, vol. 32, no. 7, pp. 454-466, July 2006.

[5] 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," *Proc. 13th Conf. Object-Oriented Programming Systems, Languages and Applications*, pp. 271-283, 1998.

[6] T.J. Biggerstaff, B.G. Mitbander, and D. Webster, "The Concept Assignment Problem in Program Understanding," *Proc. 15th Int'l Conf. Software Eng.*, pp. 482-498, 1993.

[7] T. Ball, "The Concept of Dynamic Analysis," *Proc. Seventh European Software Eng. Conf. and ACM SIGSOFT Symp. Foundations of Software Eng.*, pp. 216-234, 1999.

[8] J. Andrews, "Testing Using Log File Analysis: Tools, Methods, and Issues," *Proc. 13th Int'l Conf. Automated Software Eng.*, pp. 157-166, 1997.

[9] A.W. Biermann, "On the Inference of Turing Machines from Sample Computations," *Artificial Intelligence*, vol. 3, nos. 1-3, pp. 181-198, 1972.

[10] M.F. Kleyn and P.C. Gingrich, "Graphtrace—Understanding Object-Oriented Systems Using Concurrently Animated Views," *Proc. Third Conf. Object-Oriented Programming Systems, Languages, and Applications*, pp. 191-205, 1988.

[11] W. De Pauw, R. Helm, D. Kimelman, and J.M. Vlissides, "Visualizing the Behavior of Object-Oriented Systems," *Proc. Eighth Conf. Object-Oriented Programming Systems, Languages, and Applications*, pp. 326-337, 1993.

[12] W. De Pauw, D. Kimelman, and J.M. Vlissides, "Modeling Object-Oriented Program Execution," *Proc. European Object-Oriented Programming Conf.*, pp. 163-182, 1994.

[13] W. De Pauw, D. Lorenz, J. Vlissides, and M. Wegman, "Execution Patterns in Object-Oriented Visualization," *Proc. Fourth USENIX Conf. Object-Oriented Technologies and Systems*, pp. 219-234, 1998.

[14] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1995.

[15] N. Wilde and M.C. Scully, "Software Reconnaissance: Mapping Program Features to Code," *J. Software Maintenance: Research and Practice*, vol. 7, no. 1, pp. 49-62, 1995.

[16] N. Wilde, M. Buckellew, H. Page, V. Rajlich, and L. Pounds, "A Comparison of Methods for Locating Features in Legacy Software," *J. Systems and Software*, vol. 65, no. 2, pp. 105-114, 2003.

[17] N. Wilde, M. Buckellew, H. Page, and V. Rajlich, "A Case Study of Feature Location in Unstructured Legacy Fortran Code," *Proc. Fifth European Conf. Software Maintenance and Reeng.*, pp. 68-76, 2001.

[18] N. Wilde and C. Casey, "Early Field Experience with the Software Reconnaissance Technique for Program Comprehension," *Proc. Int'l Conf. Software Maintenance*, pp. 312-318, 1996.

[19] D.B. Lange and Y. Nakamura, "Interactive Visualization of Design Patterns Can Help in Framework Understanding," *Proc. 10th Conf. Object-Oriented Programming Systems, Languages, and Applications*, pp. 342-357, 1995.

[20] D.B. Lange and Y. Nakamura, "Program Explorer: A Program Visualizer for C++," *Proc. First USENIX Conf. Object-Oriented Technologies and Systems*, pp. 39-54, 1995.

[21] K. Koskimies and H. Mössenböck, "Scene: Using Scenario Diagrams and Active Text for Illustrating Object-Oriented Programs," *Proc. 18th Int'l Conf. Software Eng.*, pp. 366-375, 1996.

[22] M. Sefika, A. Sane, and R.H. Campbell, "Architecture-Oriented Visualization," *Proc. 11th Conf. Object-Oriented Programming Systems, Languages, and Applications*, pp. 389-405, 1996.

[23] D.F. Jerding and S. Rugaber, "Using Visualization for Architectural Localization and Extraction," *Proc. Fourth Working Conf. Reverse Eng.*, pp. 56-65, 1997.

[24] D.F. Jerding, J.T. Stasko, and T. Ball, "Visualizing Interactions in Program Executions," *Proc. 19th Int'l Conf. Software Eng.*, pp. 360-370, 1997.

[25] D.F. Jerding and J.T. Stasko, "The Information Mural: A Technique for Displaying and Navigating Large Information Spaces," *IEEE Trans. Visualization and Computer Graphics*, vol. 4, no. 3, pp. 257-271, July-Sept. 1998.

[26] G.C. Murphy, D. Notkin, and K.J. Sullivan, "Software Reflexion Models: Bridging the Gap between Design and Implementation," *IEEE Trans. Software Eng.*, vol. 27, no. 4, pp. 364-380, Apr. 2001.

[27] T. Richner and S. Ducasse, "Using Dynamic Information for the Iterative Recovery of Collaborations and Roles," *Proc. 18th Int'l Conf. Software Maintenance*, pp. 34-43, 2002.

[28] W. De Pauw, E. Jensen, N. Mitchell, G. Sevitsky, J.M. Vlissides, and J. Yang, "Visualizing the Execution of Java Programs," *Proc. ACM 2001 Symp. Software Visualization*, pp. 151-162, 2001.

[29] T. Systä, K. Koskimies, and H.A. Müller, "Shimba: An Environment for Reverse Engineering Java Software Systems," *Software, Practice and Experience*, vol. 31, no. 4, pp. 371-394, 2001.

[30] L.C. Briand, Y. Labiche, and J. Leduc, "Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software," *IEEE Trans. Software Eng.*, vol. 32, no. 9, pp. 642-663, Sept. 2006.

[31] S.P. Reiss and M. Renieris, "Encoding Program Executions," *Proc. 23rd Int'l Conf. Software Eng.*, pp. 221-230, 2001.

[32] A. Hamou-Lhadj and T.C. Lethbridge, "A Survey of Trace Exploration Tools and Techniques," *Proc. 2004 Conf. the Centre for Advanced Studies on Collaborative Research*, pp. 42-55, 2004.

[33] A. Hamou-Lhadj and T.C. Lethbridge, "Summarizing the Content of Large Traces to Facilitate the Understanding of the Behaviour of a Software System," *Proc. 14th Int'l Conf. Program Comprehension*, pp. 181-190, 2006.

[34] A. Hamou-Lhadj, T.C. Lethbridge, and L. Fu, "Challenges and Requirements for an Effective Trace Exploration Tool," *Proc. 12th Int'l Workshop Program Comprehension*, pp. 70-78, 2004.

[35] D. Heuzeroth, T. Holl, and W. Löwe, "Combining Static and Dynamic Analyses to Detect Interaction Patterns," *Proc. Sixth Int'l Conf. Integrated Design and Process Technology*, 2002.

[36] D. Heuzeroth, T. Holl, G. Högström, and W. Löwe, "Automatic Design Pattern Detection," *Proc. 11th Int'l Workshop Program Comprehension*, pp. 94-103, 2003.

[37] C. Riva and J.V. Rodriguez, "Combining Static and Dynamic Views for Architecture Reconstruction," *Proc. Sixth European Conf. Software Maintenance and Reeng.*, pp. 47-55, 2002.

[38] C. Riva and Y. Yang, "Generation of Architectural Documentation Using XML," *Proc. Ninth Working Conf. Reverse Eng.*, pp. 161-179, 2002.

[39] H. Yan, D. Garlan, B.R. Schmerl, J. Aldrich, and R. Kazman, "Discotect: A System for Discovering Architectures from Running Systems," *Proc. 26th Int'l Conf. Software Eng.*, pp. 470-479, 2004.

[40] J. Koskinen, M. Kettunen, and T. Systä, "Profile-Based Approach to Support Comprehension of Software Behavior," *Proc. 14th Int'l Conf. Program Comprehension*, pp. 212-224, 2006.

[41] J.E. Cook and Z. Du, "Discovering Thread Interactions in a Concurrent System," *J. Systems and Software*, vol. 77, no. 3, pp. 285-297, 2005.

[42] J. Quante and R. Koschke, "Dynamic Protocol Recovery," *Proc. 14th Working Conf. Reverse Eng.*, pp. 219-228, 2007.

[43] D. Lo, S. Khoo, and C. Liu, "Mining Temporal Rules for Software Maintenance," *J. Software Maintenance and Evolution: Research and Practice,* vol. 20, no. 4, pp. 227-247, 2008.

[44] N. Walkinshaw, K. Bogdanov, M. Holcombe, and S. Salahuddin, "Reverse Engineering State Machines by Interactive Grammar Inference," *J. Software Maintenance and Evolution: Research and Practice,* vol. 20, no. 4, pp. 269-290, 2008.

[45] O. Greevy, S. Ducasse, and T. Gîrba, "Analyzing Feature Traces to Incorporate the Semantics of Change in Software Evolution Analysis," *Proc. 21st Int'l Conf. Software Maintenance,* pp. 347-356, 2005.

[46] O. Greevy, S. Ducasse, and T. Gîrba, "Analyzing Software Evolution through Feature Views," *J. Software Maintenance and Evolution: Research and Practice,* vol. 18, no. 6, pp. 425-456, 2006.

[47] J. Kothari, T. Denton, A. Shokoufandeh, and S. Mancoridis, "Reducing Program Comprehension Effort in Evolving Software by Recognizing Feature Implementation Convergence," *Proc. 15th Int'l Conf. Program Comprehension,* pp. 17-26, 2007.

[48] W.E. Wong, S.S. Gokhale, and J.R. Horgan, "Quantifying the Closeness between Program Components and Features," *J. Systems and Software,* vol. 54, no. 2, pp. 87-98, 2000.

[49] T. Eisenbarth, R. Koschke, and D. Simon, "Locating Features in Source Code," *IEEE Trans. Software Eng.,* vol. 29, no. 3, pp. 210-224, Mar. 2003.

[50] G. Antoniol and Y.-G. Guéhéneuc, "Feature Identification: An Epidemiological Metaphor," *IEEE Trans. Software Eng.,* vol. 32, no. 9, pp. 627-641, Sept. 2006.

[51] 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.,* vol. 33, no. 6, pp. 420-432, June 2007.

[52] M.J. Pacione, M. Roper, and M. Wood, "Comparative Evaluation of Dynamic Visualisation Tools," *Proc. 10th Working Conf. Reverse Eng.,* pp. 80-89, 2003.

[53] O. Greevy, "Enriching Reverse Engineering with Feature Analysis," PhD thesis, Univ. Bern, 2007.

[54] S.P. Reiss, "Visual Representations of Executing Programs," *J. Visual Languages and Computing,* vol. 18, no. 2, pp. 126-148, 2007.

[55] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in Software Engineering: A Systematic Literature Review," *Information and Software Technology,* vol. 50, nos. 9/10, pp. 860-878, 2008.

[56] T. Dybå and T. Dingsøyr, "Empirical Studies of Agile Software Development: A Systematic Review," *Information and Software Technology,* vol. 50, nos. 9/10, pp. 833-859, 2008.

[57] 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. Systems and Software,* vol. 80, no. 4, pp. 571-583, 2007.

[58] 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.,* vol. 31, no. 9, pp. 733-753, Sept. 2005.

[59] B.A. Kitchenham, "Procedures for Performing Systematic Reviews," Technical Report TR/SE-0401, Keele Univ. and Technical Report 0400011T.1, Nat'l ICT Australia, 2004.

[60] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A Systematic Survey of Program Comprehension through Dynamic Analysis," Technical Report TUD-SERG-2008-033, Delft Univ. of Technology, 2008.

[61] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature Location via Information Retrieval Based Filtering of a Single Scenario Execution Trace," *Proc. 22nd Int'l Conf. Automated Software Eng.,* pp. 234-243, 2007.

[62] A. Zaidman and S. Demeyer, "Managing Trace Data Volume through a Heuristical Clustering Process Based on Event Execution Frequency," *Proc. Eighth European Conf. Software Maintenance and Reeng.,* pp. 329-338, 2004.

[63] M.D. Ernst, "Static and Dynamic Analysis: Synergy and Duality," *Proc. First ICSE Int'l Workshop Dynamic Analysis,* pp. 25-28, 2003.

[64] T. Israr, M. Woodside, and G. Franks, "Interaction Tree Algorithms to Extract Effective Architecture and Layered Performance Models from Traces," *J. Systems and Software,* vol. 80, no. 4, pp. 474-492, 2007.

[65] M. Benedikt, J. Freire, and P. Godefroid, "VeriWeb: Automatically Testing Dynamic Web Sites," *Proc. 11th Int'l Conf. World Wide Web,* 2002.

[66] A. Mesbah, E. Bozdag, and A. van Deursen, "Crawling Ajax by Inferring User Interface State Changes," *Proc. Eighth Int'l Conf. Web Eng.,* pp. 122-134, 2008.

[67] N. Gold, A. Mohan, C. Knight, and M. Munro, "Understanding Service-Oriented Software," *IEEE Software,* vol. 21, no. 2, pp. 71-77, Mar./Apr. 2004.

[68] J. Moe and D.A. Carr, "Understanding Distributed Systems via Execution Trace Data," *Proc. Ninth Int'l Workshop Program Comprehension,* pp. 60-67, 2001.

[69] V. Pankratius, C. Schaefer, A. Jannesari, and W.F. Tichy, "Software Engineering for Multicore Systems: An Experience Report," *Proc. First ICSE Int'l Workshop Multicore Software Eng.,* 2008.

[70] A. Zaidman, S. Demeyer, B. Adams, K. De Schutter, G. Hoffman, and B. De Ruyck, "Regaining Lost Knowledge through Dynamic Analysis and Aspect Orientation," *Proc. 10th European Conf. Software Maintenance and Reeng.,* pp. 91-102, 2006.

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

[72] D. Röthlisberger, O. Greevy, and O. Nierstrasz, "Exploiting Runtime Information in the IDE," *Proc. 16th Int'l Conf. Program Comprehension,* pp. 63-72, 2008.

[73] S.P. Reiss, "Visualizing Java in Action," *Proc. ACM 2003 Symp. Software Visualization,* pp. 57-65, 2003.

[74] S. Joshi and A. Orso, "SCARPE: A Technique and Tool for Selective Record and Replay of Program Executions," *Proc. 23rd Int'l Conf. Software Maintenance,* pp. 234-243, 2007.

[75] G. Xu, A. Rountev, Y. Tang, and F. Qin, "Efficient Checkpointing of Java Software Using Context-Sensitive Capture and Replay," *Proc. 15th European Software Eng. Conf. and ACM SIGSOFT Symp. the Foundations of Software Eng.,* pp. 85-94, 2007.

[76] A. Zaidman, "Scalability Solutions for Program Comprehension through Dynamic Analysis," PhD thesis, Univ. of Antwerp, 2006.

[77] 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. Software Maintenance and Evolution: Research and Practice,* vol. 20, no. 4, pp. 291-315, 2008.

[78] J. Quante, "Do Dynamic Object Process Graphs Support Program Understanding?—A Controlled Experiment," *Proc. 16th Int'l Conf. Program Comprehension,* pp. 73-82, 2008.

[79] S. Simmons, D. Edwards, N. Wilde, J. Homan, and M. Groble, "Industrial Tools for the Feature Location Problem: An Exploratory Study," *J. Software Maintenance and Evolution: Research and Practice,* vol. 18, no. 6, pp. 457-474, 2006.

[80] 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," *Proc. 16th Int'l Conf. Program Comprehension,* pp. 53-62, 2008.

[81] B. Cornelissen, L. Moonen, and A. Zaidman, "An Assessment Methodology for Trace Reduction Techniques," *Proc. 24th Int'l Conf. Software Maintenance,* pp. 107-116, 2008.

[82] S.E. Sim, S.M. Easterbrook, and R.C. Holt, "Using Benchmarking to Advance Research: A Challenge to Software Engineering," *Proc. 25th Int'l Conf. Software Eng.,* pp. 74-83, 2003.

[83] M. Staples and M. Niazi, "Experiences Using Systematic Review Guidelines," *J. Systems and Software,* vol. 80, no. 9, pp. 1425-1437, 2007.

[84] G. Antoniol and M. Di Penta, "A Distributed Architecture for Dynamic Analyses on User-Profile Data," *Proc. Eighth European Conf. Software Maintenance and Reeng.,* pp. 319-328, 2004.

[85] G. Antoniol, M. Di Penta, and M. Zazzara, "Understanding Web Applications through Dynamic Analysis," *Proc. 12th Int'l Workshop Program Comprehension,* pp. 120-131, 2004.

[86] T. Ball, "Software Visualization in the Large," *Computer,* vol. 29, no. 4, pp. 33-43, Apr. 1996.

[87] J. Bohnet and J. Döllner, "Visual Exploration of Function Call Graphs for Feature Location in Complex Software Systems," *Proc. ACM 2006 Symp. Software Visualization,* pp. 95-104, 2006.

[88] D. Bojic and D.M. Velasevic, "A Use-Case Driven Method of Architecture Recovery for Program Understanding and Reuse Reengineering," *Proc. Fourth European Conf. Software Maintenance and Reeng.,* pp. 23-32, 2000.

[89] A. Chan, R. Holmes, G.C. Murphy, and A.T.T. Ying, "Scaling an Object-Oriented System Execution Visualizer through Sampling," *Proc. 11th Int'l Workshop Program Comprehension,* pp. 237-244, 2003.

[90] B. Cornelissen, A. van Deursen, L. Moonen, and A. Zaidman, “Visualizing Testsuites to Aid in Software Understanding,” *Proc. 11th European Conf. Software Maintenance and Reeng.*, pp. 213-222, 2007.

[91] B. Cornelissen, A. Zaidman, D. Holten, L. Moonen, A. van Deursen, and J.J. van Wijk, “Execution Trace Analysis through Massive Sequence and Circular Bundle Views,” *J. Systems and Software*, vol. 81, no. 11, pp. 2252-2268, 2008.

[92] A.R. Dalton and J.O. Hallstrom, “A Toolkit for Visualizing the Runtime Behavior of TinyOS Applications,” *Proc. 15th Int'l Conf. Program Comprehension*, pp. 43-52, 2008.

[93] J. Deprez and A. Lakhotia, “A Formalism to Automate Mapping from Program Features to Code,” *Proc. Eighth Int'l Workshop Program Comprehension*, pp. 69-78, 2000.

[94] S. Ducasse, M. Lanza, and R. Bertuli, “High-Level Polymetric Views of Condensed Run-Time Information,” *Proc. Eighth European Conf. Software Maintenance and Reeng.*, pp. 309-318, 2004.

[95] D. Edwards, S. Simmons, and N. Wilde, “An Approach to Feature Location in Distributed Systems,” *J. Systems and Software*, vol. 79, no. 1, pp. 457-474, 2006.

[96] A.D. Eisenberg and K. De Volder, “Dynamic Feature Traces: Finding Features in Unfamiliar Code,” *Proc. 21st Int'l Conf. Software Maintenance*, pp. 337-346, 2005.

[97] M. El-Ramly, E. Stroulia, and P.G. Sorenson, “From Run-Time Behavior to Usage Scenarios: An Interaction-Pattern Mining Approach,” *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 315-324, 2002.

[98] M. Fischer, J. Oberleitner, H. Gall, and T. Gschwind, “System Evolution Tracking through Execution Trace Analysis,” *Proc. 13th Int'l Workshop Program Comprehension*, pp. 237-246, 2005.

[99] M. Fisher II, S.G. Elbaum, and G. Rothermel, “Dynamic Characterization of Web Application Interfaces,” *Proc. 10th Int'l Conf. Fundamental Approaches to Software Eng.*, pp. 260-275, 2007.

[100] J. Gargiulo and S. Mancoridis, “Gadget: A Tool for Extracting the Dynamic Structure of Java Programs,” *Proc. 13th Int'l Conf. Software Eng. and Knowledge Eng.*, pp. 244-251, 2001.

[101] P.V. Gestwicki and B. Jayaraman, “Methodology and Architecture of JIVE,” *Proc. ACM 2005 Symp. Software Visualization*, pp. 95-104, 2005.

[102] O. Greevy, M. Lanza, and C. Wysseier, “Visualizing Live Software Systems in 3D,” *Proc. ACM 2006 Symp. Software Visualization*, pp. 47-56, 2006.

[103] T. Gschwind, J. Oberleitner, and M. Pinzger, “Using Run-Time Data for Program Comprehension,” *Proc. 11th Int'l Workshop Program Comprehension*, pp. 245-250, 2003.

[104] Y.-G. Guéhéneuc, R. Douence, and N. Jussien, “No Java without Caffeine: A Tool for Dynamic Analysis of Java Programs,” *Proc. 17th Int'l Conf. Automated Software Eng.*, pp. 117-126, 2002.

[105] Y.-G. Guéhéneuc, “A Reverse Engineering Tool for Precise Class Diagrams,” *Proc. 2004 Conf. Centre for Advanced Studies on Collaborative Research*, pp. 28-41, 2004.

[106] S.A. Hendrickson, E.M. Dashofy, and R.N. Taylor, “An Architecture-Centric Approach for Tracing, Organizing, and Understanding Events in Event-Based Software Architectures,” *Proc. 13th Int'l Workshop Program Comprehension*, pp. 227-236, 2005.

[107] H. Huang, S. Zhang, J. Cao, and Y. Duan, “A Practical Pattern Recovery Approach Based on Both Structural and Behavioral Analysis,” *J. Systems and Software*, vol. 75, nos. 1/2, pp. 69-87, 2005.

[108] J. Jiang, J. Koskinen, A. Ruokonen, and T. Systä, “Constructing Usage Scenarios for API Redocumentation,” *Proc. 15th Int'l Conf. Program Comprehension*, pp. 259-264, 2007.

[109] M. Jiang, M. Groble, S. Simmons, D. Edwards, and N. Wilde, “Software Feature Understanding in an Industrial Setting,” *Proc. 22nd Int'l Conf. Software Maintenance*, pp. 66-67, 2006.

[110] Z.M. Jiang, A. Hassan, G. Hamann, and P. Flora, “An Automated Approach for Abstracting Execution Logs to Execution Events,” *J. Software Maintenance and Evolution: Research and Practice*, vol. 20, no. 4, pp. 249-267, 2008.

[111] P. Kelsen, “A Simple Static Model for Understanding the Dynamic Behavior of Programs,” *Proc. 12th Int'l Workshop Program Comprehension*, pp. 46-51, 2004.

[112] R. Kollmann and M. Gogolla, “Capturing Dynamic Program Behaviour with UML Collaboration Diagrams,” *Proc. Fifth European Conf. Software Maintenance and Reeng.*, pp. 58-67, 2001.

[113] B. Korel and J. Rilling, “Program Slicing in Understanding of Large Programs,” *Proc. Sixth Int'l Workshop Program Comprehension*, pp. 145-152, 1998.

[114] R. Koschke and J. Quante, “On Dynamic Feature Location,” *Proc. 20th Int'l Conf. Automated Software Eng.*, pp. 86-95, 2005.

[115] A. Kuhn and O. Greevy, “Exploiting the Analogy between Traces and Signal Processing,” *Proc. 22nd Int'l Conf. Software Maintenance*, pp. 320-329, 2006.

[116] D. Lange and Y. Nakamura, “Object-Oriented Program Tracing and Visualization,” *Computer*, vol. 30, no. 5, pp. 63-70, May 1997.

[117] D.R. Licata, C.D. Harris, and S. Krishnamurthi, “The Feature Signatures of Evolving Programs,” *Proc. 18th Int'l Conf. Automated Software Eng.*, pp. 281-285, 2003.

[118] A. Lienhard, O. Greevy, and O. Nierstrasz, “Tracking Objects to Detect Feature Dependencies,” *Proc. 15th Int'l Conf. Program Comprehension*, pp. 59-68, 2007.

[119] D. Lo and S. Khoo, “QUARK: Empirical Assessment of Automaton-Based Specification Miners,” *Proc. 13th Working Conf. Reverse Eng.*, pp. 51-60, 2006.

[120] D. Lo and S. Khoo, “SMArTIC: Towards Building an Accurate, Robust and Scalable Specification Miner,” *Proc. 14th European Software Eng. Conf. and ACM SIGSOFT Symp. the Foundations of Software Eng.*, pp. 265-275, 2006.

[121] G.A. Di Lucca, A.R. Fasolino, P. Tramontana, and U. de Carlini, “Abstracting Business Level UML Diagrams from Web Applications,” *Proc. Fifth Int'l Workshop Web Site Evolution*, pp. 12-19, 2003.

[122] K. Lukoit, N. Wilde, S. Stowell, and T. Hennessey, “TraceGraph: Immediate Visual Location of Software Features,” *Proc. 16th Int'l Conf. Software Maintenance*, pp. 33-39, 2000.

[123] B.A. Malloy and J.F. Power, “Exploiting UML Dynamic Object Modeling for the Visualization of C++ Programs,” *Proc. ACM 2005 Symp. Software Visualization*, pp. 105-114, 2005.

[124] L. Martin, A. Giesl, and J. Martin, “Dynamic Component Program Visualization,” *Proc. Ninth Working Conf. Reverse Eng.*, pp. 289-298, 2002.

[125] J. Moe and K. Sandahl, “Using Execution Trace Data to Improve Distributed Systems,” *Proc. 18th Int'l Conf. Software Maintenance*, pp. 640-648, 2002.

[126] R. Oechsle and T. Schmitt, “JAVAVIS: Automatic Program Visualization with Object and Sequence Diagrams Using the Java Debug Interface (JDI),” *Proc. ACM 2001 Symp. Software Visualization*, pp. 176-190, 2001.

[127] A. Orso, J.A. Jones, and M.J. Harrold, “Visualization of Program-Execution Data for Deployed Software,” *Proc. ACM 2003 Symp. Software Visualization*, pp. 67-76, 2003.

[128] M.J. Pacione, M. Roper, and M. Wood, “A Novel Software Visualisation Model to Support Software Comprehension,” *Proc. 11th Working Conf. Reverse Eng.*, pp. 70-79, 2004.

[129] W. De Pauw, S. Krasikov, and J.F. Morar, “Execution Patterns for Visualizing Web Services,” *Proc. ACM 2006 Symp. Software Visualization*, pp. 37-45, 2006.

[130] S. Pheng and C. Verbrugge, “Dynamic Data Structure Analysis for Java Programs,” *Proc. 14th Int'l Conf. Program Comprehension*, pp. 191-201, 2006.

[131] L. Qingshan, H. Chu, S. Hu, P. Chen, and Z. Yun, “Architecture Recovery and Abstraction from the Perspective of Processes,” *Proc. 12th Working Conf. Reverse Eng.*, pp. 57-66, 2005.

[132] S.P. Reiss, “Event-Based Performance Analysis,” *Proc. 11th Int'l Workshop Program Comprehension*, pp. 74-83, 2003.

[133] S.P. Reiss, “Visualizing Program Execution Using User Abstractions,” *Proc. ACM 2006 Symp. Software Visualization*, pp. 125-134, 2006.

[134] M. Renieris and S.P. Reiss, “Almost: Exploring Program Traces,” *Proc. 1999 Workshop New Paradigms in Information Visualization and Manipulation*, pp. 70-77, 1999.

[135] J. Rilling, “Maximizing Functional Cohesion of Comprehension Environments by Integrating User and Task Knowledge,” *Proc. Eighth Working Conf. Reverse Eng.*, pp. 157-165, 2001.

[136] H. Ritsch and H.M. Sneed, “Reverse Engineering Programs via Dynamic Analysis,” *Proc. First Working Conf. Reverse Eng.*, pp. 192-201, 1993.

[137] C. Riva, “Reverse Architecting: An Industrial Experience Report,” *Proc. Seventh Working Conf. Reverse Eng.*, pp. 42-50, 2000.

[138] A. Rohatgi, A. Hamou-Lhadj, and J. Rilling, “An Approach for Mapping Features to Code Based on Static and Dynamic Analysis,” *Proc. 16th Int'l Conf. Program Comprehension*, pp. 236-241, 2008.

[139] C. De Roover, I. Michiels, K. Gybels, K. Gybels, and T. D'Hondt, "An Approach to High-Level Behavioral Program Documentation Allowing Lightweight Verification," *Proc. 14th Int'l Conf. Program Comprehension,* pp. 202-211, 2006.

[140] H. Safyallah and K. Sartipi, "Dynamic Analysis of Software Systems Using Execution Pattern Mining," *Proc. 14th Int'l Conf. Program Comprehension,* pp. 84-88, 2006.

[141] M. Salah and S. Mancoridis, "Toward an Environment for Comprehending Distributed Systems," *Proc. 10th Working Conf. Reverse Eng.,* pp. 238-247, 2003.

[142] M. Salah and S. Mancoridis, "A Hierarchy of Dynamic Software Views: From Object-Interactions to Feature-Interactions," *Proc. 20th Int'l Conf. Software Maintenance,* pp. 72-81, 2004.

[143] M. Salah, S. Mancoridis, G. Antoniol, and M. Di Penta, "Scenario-Driven Dynamic Analysis for Comprehending Large Software Systems," *Proc. 10th European Conf. Software Maintenance and Reeng.,* pp. 71-80, 2006.

[144] K. Sartipi and N. Dezhkam, "An Amalgamated Dynamic and Static Architecture Reconstruction Framework to Control Component Interactions," *Proc. 14th Working Conf. Reverse Eng.,* pp. 259-268, 2007.

[145] M. Shevertalov and S. Mancoridis, "A Reverse Engineering Tool for Extracting Protocols of Networked Applications," *Proc. 14th Working Conf. Reverse Eng.,* pp. 229-238, 2007.

[146] M. Smit, E. Stroulia, and K. Wong, "Use Case Redocumentation from Gui Event Traces," *Proc. 12th European Conf. Software Maintenance and Reeng.,* pp. 263-268, 2008.

[147] T.S. Souder, S. Mancoridis, and M. Salah, "Form: A Framework for Creating Views of Program Executions," *Proc. 17th Int'l Conf. Software Maintenance,* pp. 612-620, 2001.

[148] F.C. de Sousa, N.C. Mendonça, S. Uchitel, and J. Kramer, "Detecting Implied Scenarios from Execution Traces," *Proc. 14th Working Conf. Reverse Eng.,* pp. 50-59, 2007.

[149] E. Stroulia, M. El-Ramly, L. Kong, P.G. Sorenson, and B. Matichuk, "Reverse Engineering Legacy Interfaces: An Interaction-Driven Approach," *Proc. Sixth Working Conf. Reverse Eng.,* pp. 292-302, 1999.

[150] P. Tonella and A. Potrich, "Static and Dynamic C++ Code Analysis for the Recovery of the Object Diagram," *Proc. 18th Int'l Conf. Software Maintenance,* pp. 54-63, 2002.

[151] R.J. Walker, G.C. Murphy, J. Steinbok, and M.P. Robillard, "Efficient Mapping of Software System Traces to Architectural Views," *Proc. 2000 Conf. the Centre for Advanced Studies on Collaborative Research,* pp. 12-21, 2000.

[152] L. Wang, J.R. Cordy, and T.R. Dean, "Enhancing Security Using Legality Assertions," *Proc. 12th Working Conf. Reverse Eng.,* pp. 35-44, 2005.

[153] W.E. Wong and S.S. Gokhale, "Static and Dynamic Distance Metrics for Feature-Based Code Analysis," *J. Systems and Software,* vol. 74, no. 3, pp. 283-295, 2005.

[154] A. Zaidman and S. Demeyer, "Automatic Identification of Key Classes in a Software System Using Webmining Techniques," *J. Software Maintenance and Evolution: Research and Practice,* vol. 20, no. 6, pp. 387-417, 2008.

Bas Cornelissen received the MSc degree in computer science from the University of Amsterdam in 2005. Since June 2005, he has been working as a graduate student at the Delft University of Technology. His research interests include program comprehension and reverse engineering, with a strong emphasis on the use of dynamic analysis. He is a student member of the IEEE and the IEEE Computer Society.

Andy Zaidman received the MSc and PhD degrees in computer science from the University of Antwerp, Belgium, in 2002 and 2006, respectively, after which he became a postdoctoral researcher at the Delft University of Technology, where he is now an assistant professor. His main research interests are reverse engineering with the help of dynamic analysis, program comprehension, software repository mining, and software testing. He is the founder and an organizer of the International Workshop on Program Comprehension through Dynamic Analysis (PCODA) series which started in 2005. He was the general chair of the 15th Working Conference on Reverse Engineering (WCRE '08) held in Antwerp, Belgium, and will be the program cochair for WCRE '09. He is a member of the IEEE Computer Society.

Arie van Deursen received the MSc degree in computer science from the Vrije Universiteit, Amsterdam, in 1990, and the PhD degree from the University of Amsterdam in 1994. He is a full professor at the Delft University of Technology, where he heads the Software Engineering Research Group. From 1996 to 2006, he was a research leader at CWI, the Dutch National Institute for Research in Mathematics and Computer Science. His research interests include reverse engineering, program comprehension, and software architecture. He was a program chair of the Working Conference on Reverse Engineering (WCRE) in 2002 and 2003, and served on numerous program committees in the areas of software evolution, maintenance, and software engineering in general. He is a member of the editorial board of the *Empirical Software Engineering Journal*. He is a member of the IEEE Computer Society.

Leon Moonen received the MSc degree (cum laude) in computer science and the PhD degree in computer science from the University of Amsterdam in 1996 and 2002, respectively. He is a research scientist at the Simula Research Laboratory, Norway. His research focuses on the design and development of techniques and tools for the exploration, evolution, and assessment of industrial-size software systems. His research interests include automated software inspection, software repository mining, reverse engineering, program analysis, and program comprehension. He publishes regularly and serves on organizing, steering, and program committees of international conferences and workshops on reverse engineering, source code analysis, software maintenance, program understanding, aspect mining, and software security. He is a member of the IEEE Computer Society and the ACM.

Rainer Koschke received the doctoral degree in computer science from the University of Stuttgart, Germany. He is a full professor of software engineering at the University of Bremen in Germany. His research interests are primarily in the fields of software evolution and maintenance and program analyzes. His current research includes architecture recovery, feature location, program analyzes, software clone detection, and reverse engineering. He is a member of the IEEE Computer Society and the Gesellschaft für Informatik e.V. (GI; German Society of Computer Science).