Performance Analysis for Object-Oriented Software: A Systematic Mapping

David Maplesden, Ewan Tempero, *Member, IEEE*, John Hosking, *Member, IEEE*, and John C. Grundy, *Member, IEEE*

Abstract—Performance is a crucial attribute for most software, making performance analysis an important software engineering task. The difficulty is that modern applications are challenging to analyse for performance. Many profiling techniques used in real-world software development struggle to provide useful results when applied to large-scale object-oriented applications. There is a substantial body of research into software performance generally but currently there exists no survey of this research that would help identify approaches useful for object-oriented software. To provide such a review we performed a systematic mapping study of empirical performance analysis approaches that are applicable to object-oriented software. Using keyword searches against leading software engineering research databases and manual searches of relevant venues we identified over 5,000 related articles published since January 2000. From these we systematically selected 253 applicable articles and categorised them according to ten facets that capture the intent, implementation and evaluation of the approaches. Our mapping study results allow us to highlight the main contributions of the existing literature and identify areas where there are interesting opportunities. We also find that, despite the research including approaches specifically aimed at object-oriented software, there are significant challenges in providing actionable feedback on the performance of large-scale object-oriented applications.

Index Terms—Systematic review, survey, performance, object-oriented

1 Introduction

Software performance has been of interest to researchers and practitioners since the earliest days of computing but it is still highly relevant today. The performance of object-oriented software is of particular interest because object-oriented languages and techniques are ubiquitous in industry, underlined by the popularity of object-oriented languages such as Java, C++ and C#. There is a significant collection of published literature relating to software performance, a large subset of which is applicable to object-oriented software. However, to the best of our knowledge, there exists no previous attempt to survey the literature in this field. Such a survey would be a valuable contribution for researchers and practitioners looking to understand the existing research in the field either for the purposes of leveraging or contributing to that research. To that end we have undertaken an in-depth systematic mapping study, a form of systematic literature review (SLR) [1], of the field.

Software efficiency is still important for today's applications despite the prodigious improvement in hardware performance over the last three decades. There are several trends in application development that contribute to a ongoing need for efficient software. The size and complexity of software has increased at a similar or even greater rate than hardware has advanced [2]. There are increasing numbers of mobile applications being created which must run on devices with limited resources. Also many applications are delivered using cloud deployment models where running costs are directly impacted by software efficiency. The challenge is that the growing scale and complexity of the software under development means that analysing and improving the performance of these systems has become increasingly difficult.

There are specific challenges for object-oriented applications because they have characteristics that make performance analysis difficult. Following object-oriented principles tends to lead to applications with inter-procedural rather than intra-procedural control flow and a great number of methods. Additionally many object-oriented methodologies focus on developer productivity, producing maintainable and flexible software, and promoting componentisation and reuse. As a result most applications are built from reusable generalised frameworks and leverage established design patterns, making them very layered and complex. This approach means that the handling of even the simplest request in these framework-based applications goes through many layers and will require hundreds, maybe thousands, of method calls to complete [3]. This excessive activity to achieve seemingly simple results is a problem that has become known as *runtime bloat* [4], and it has led many large scale object-oriented applications to suffer from chronic performance problems [5].

- *D. Maplesden and E. Tempero are with the Department of Computer Science, University of Auckland, Private Bag 92019, Auckland 1142, New Zealand. E-mail: dmap001@aucklanduni.ac.nz, e.tempero@cs.auckland.ac.nz.*
- *J. Hosking is with the Faculty of Science, University of Auckland, Private Bag 92019, Auckland 1142, New Zealand. E-mail: j.hosking@auckland.ac.nz.*
- *J.C. Grundy is with the Faculty of Information and Communication Technologies, Swinburne University of Technology, PO Box 218, Hawthorn, Vic. 3122, Australia. E-mail: jgrundy@swin.edu.au.*

Manuscript received 22 May 2014; revised 15 Dec. 2014; accepted 21 Jan. 2015. Date of publication 26 Jan. 2015; date of current version 17 July 2015. Recommended for acceptance by M. Woodside.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2015.2396514

The scope of this survey then is performance analysis approaches that are *applicable* to object-oriented software. We have not limited the scope to just approaches specific to object-oriented software but have also included approaches that are applicable to most applications and are therefore likely to be useful when analysing a typical object-oriented application. The aim is provide a survey of techniques that are relevant both for practitioners who develop object-oriented applications as well as researchers interested in large-scale object-oriented applications and runtime bloat.

The survey covers empirical performance analysis approaches described in the literature since January 2000. We have chosen to cover empirical approaches only, and exclude model-based approaches (discussed in Section 2.1), to keep the survey to a manageable size and because empirical approaches are prevalent in industry [6]. We chose January 2000 as the start date as this represented a practical starting point for the review, it was far enough in the past to include the majority of relevant approaches but recent enough that all relevant published literature was indexed and available within the major electronic databases.

The remainder of this paper is organised as follows: in the next section we describe in more detail the domain of our survey and summarise the most closely related work that we did *not* include in our formal survey. Section 3 describes our methodology for the review, including our specific research questions and the details of the categorisation we used for the data extraction. Section 4 contains the results of our data extraction and conclusions for our research questions. Section 5 is a discussion of the results. Section 6 evaluates the threats to validity of our survey and we conclude this paper in Section 7.

2 Background and Motivation

2.1 Software Performance Engineering (SPE)

Many different approaches are used when seeking to improve system performance. These include research into microprocessor or computer architecture design (i.e. hardware performance), static optimisation techniques (usually compiler based), dynamic binary translation or dynamic optimisation systems (which look to automatically tune the executing code at runtime) and software engineering approaches (which target improvements in the software architecture or source code implementation). Generally speaking each of these approaches is complementary to the others. They tend to target different types of inefficiency, and each can produce useful performance improvements independently of the others.

The realm of software engineering approaches to improving performance is known as *software performance engineering*. Woodside et al. [6] give the following definition:

> Software performance engineering represents the entire collection of software engineering activities and related analyses used throughout the software development cycle, which are directed to meeting performance requirements.

Woodside et al. also distinguish between two distinct approaches to SPE: model-based predictive performance engineering and empirical performance analysis. Often the term software performance engineering is used to refer only to model-based approaches rather than the broader definition given by Woodside et al.

Model-based approaches, made popular by Smith [7], consist of constructing and solving theoretical models of a system to predict its performance characteristics. There are a variety of different mathematical models used to support model-based performance prediction including queueing networks, petri nets and stochastic process algebras as surveyed by Balsamo et al. [8].

By contrast empirical performance analysis approaches are based on the analysis of concrete measurements taken from running software and include the traditional application profiling and tuning approaches.

Model-based approaches and empirical analysis are generally accepted as complementary as they tend to target different categories of performance problem i.e. structural or architectural versus implementation respectively.

2.2 Challenges of Empirical Performance Analysis

Empirical performance analysis approaches record detailed data concerning the dynamic behaviour of a running system. With the increasing scale of dynamic behaviour in modern systems the amount of data being produced has become overwhelming. D'Elia et al. [9] report results for short runs of a variety of off-the-shelf applications in a typical Linux distribution, some of which yield statistics for millions of different active code paths. This amount of data is very difficult to interpret manually therefore we are interested in the form of feedback provided by empirical performance analysis approaches and the techniques they use to aid with interpreting performance data. In particular we are looking for approaches that aim to provide *actionable* feedback, information that provides specific advice on what to change, and even how to make the change to improve the performance of an application.

It is generally held that performance analysis is challenging and consequently a significant number of projects suffer from performance problems [5], [6]. Therefore we are interested in how effective a performance analysis approach is at helping to improve performance. To that end we are interested in how the approaches described in the literature are evaluated.

It is also relevant how practical it is to apply an approach, as this can impact the utility of the approach. In order to obtain useful results it is important that the dynamic behaviour being measured is representative of typical behaviour. Much of today's software runs in *live production environments*. These are systems (such as web applications) which are required to be always online and service users' requests in real-time. Approaches that can be directly used in these live production systems have an advantage because they can measure the target behaviour directly. Approaches that cannot be used in production environments, because they rely on specialised customisations or impose too much overhead, must instead use a representative test system to gather their measurements. This can be a challenge because for many systems replicating realistic environment, data and load conditions is very difficult.

2.3 Related Surveys

There are three existing survey papers published since the year 2000 that discuss literature in some area of software performance [4], [8], [10]. None of the three survey papers we found were systematic literature reviews. Two were traditional expert reviews of a particular specialised field and one was a position paper that included a survey of current research into runtime bloat.

Koziolek [10] presented a survey on the performance evaluation of component-based software systems. The survey was limited to discussing approaches for component-based software, that is software based on frameworks such as Java EJB, Microsoft COM or CORBA CCM, but did cover both empirical measurement based and predictive model-based approaches. The survey contains a detailed breakdown and analysis of the literature covered with useful information and conclusions for both practitioners and researchers, however with its focus on component-based software and inclusion of model-based approaches it only covers a small subset of the research we are interested in.

Xu et al. [4] summarised the state of research into runtime bloat. The focus of this position paper was to describe software bloat and to argue why it is primarily a software engineering problem. The paper was motivated by many of the same concerns that have motivated our research. It surveys existing research into runtime bloat and outlines possible future research directions. However currently there is only a small body of work that identifies itself as investigating software bloat and therefore the paper only covers a small amount of the literature we are interested in that is applicable to object-oriented software performance more generally.

Balsamo et al. [8] surveyed model-based performance prediction in software development. The survey completes a detailed description, classification and analysis of 16 different integrated methods for model-based software performance prediction. The goal for the survey is to assess the maturity of the research in the field and point out promising research directions. The survey has a different focus from ours and they do not consider empirical performance analysis approaches.

2.4 Related Literature

Besides the related surveys described above there is a great deal of literature related to software performance we did not include in our review. Here we provide a description of the major research areas that we excluded from our survey.

As we noted in Section 2.1 our focus on empirical performance engineering has meant we have excluded literature related specifically to model-based SPE approaches. These are performance prediction approaches based on constructing and solving mathematical models of the running system. There is a large body of literature describing how to create or improve useful mathematical models, discussing new more efficient algorithms for solving the created models or describing tools and methodologies for applying the approaches in practice.

Because of our focus on object-oriented software there is literature that we excluded because it had a specific focus on some other specialised field. This included performance analysis approaches for:

- Real-time systems
- Embedded systems
- Distributed systems
- High performance computing (HPC) and massively parallel systems

Performance analysis approaches focussed in these areas are concerned with aspects of performance that are not applicable to object-oriented software generally. For example embedded systems tend to have severely constrained hardware requirements and specialised runtime environments that make performance monitoring difficult, and performance analysis for distributed systems usually focuses on remote communication patterns. We also excluded a substantial amount of research into performance for HPC and massively parallel systems that focuses on the challenges unique to supercomputing and cluster computing scenarios. This includes research into understanding and improving parallelisation and inter-node communication, computer architecture related inefficiencies such as non-uniform memory access, and approaches for processing, analysing and visualising the enormous quantities of performance data that are generated by high-end systems.

Finally we excluded a range of research that leveraged dynamic analysis of software but had goals other than performance understanding. These goals included:

- General purpose program comprehension or reverse engineering
- Automated verification or validation, defect detection
- Workload characterisation, normally for automated test workload generation
- Monitoring for intrusion detection
- Monitoring for quality of service violations or capacity planning purposes

All of these use runtime monitoring or profiling systems that are similar in nature to the dynamic data capture systems that are used for performance analysis.

3 Methodology

3.1 Overview

A systematic literature review (SLR) is described by Kitchenham & Charters [1] as *"a methodologically rigorous review of research results"*. A systematic mapping study is a form of SLR that aims to give a broader overview of a particular field. It does not evaluate the articles in as much depth as an SLR, with the advantage that a broader range of primary studies is covered [11].

We have undertaken a systematic mapping study of empirical performance analysis approaches applicable to object-oriented software. In general we followed the SLR guidelines produced by Kitchenham & Charters [1], [12] and also incorporated some of the recommendations given by Petersen et al. [11].

The high level steps in our review process were:

1) Define research questions
2) Perform manual search to pilot inclusion/exclusion criteria and generate a reference set of articles

3) Develop automated search strategy
4) Formalise review protocol
5) Conduct search for relevant studies
6) Screen and select studies for inclusion
7) Data extraction
8) Analysis

The manual search at step 2 was a necessary addition for us to the typical SLR process. As described in Section 3.3 it assisted us with the development of the inclusion criteria and the automated search phrases that were required to create the formal review protocol.

The other departure from recommended SLR procedure was that we did not perform any quality assessment on the primary studies. This is customary for mapping studies aiming to structure the literature in an entire field in order to include as much relevant literature as possible [11].

3.2 Research Questions

The focus of our systematic review is empirical software engineering approaches to performance analysis for object-oriented software. Our research questions are motivated by this focus and by the challenges in empirical performance analysis discussed in Section 2.2:

1) What approaches to empirical performance analysis have been proposed that are applicable to object-oriented software?
2) How can these approaches be characterised?
3) What form of feedback does the approach provide?
4) How practical to apply are the approaches?
   a) What is required to apply the approach?
   b) Can the approaches be applied to live production environments?
5) How are approaches evaluated and validated?

3.3 Initial Manual Search

In order to develop our review protocol we needed to develop both the automated search phrase and the inclusion criteria used to select the primary studies. To do this we undertook a manual search phase against selected venues where we could not only pilot and test our inclusion criteria but also generate a reference set of results with which to test and refine our automated searches. Ideally such a reference set would come from an independent source, such as a previously published literature review, but we had not been able to identify a suitable existing source.

We conducted our initial manual search against:

- *ICPE*—ACM/SPEC International Conference on Performance Engineering (ICPE)—2012, 2011, 2010
- *WOSP*—International Workshop on Software and Performance—2008, 2007
- *Performance Evaluation*—Journal—2012, 2011, 2010, 2009.

We then expanded our reference set by screening the papers referenced from the selected papers (reference snowballing) using our inclusion/exclusion criteria until we had an initial reference set of 46 papers.

3.4 Search Phrase Development

The search phrase was developed based on search terms extracted from the abstracts, titles and keywords from the reference set of articles and expanded with synonyms.

The development of the search phrase was challenging due to the generic nature of the obvious individual search terms: 'software', 'performance'—using these terms resulted in searches that returned many thousands of results i.e. low precision. There is also a plethora of synonyms for software performance, meaning that each individual search term has only a very low recall.

After extensive experimentation we found an effective approach was to use a search phrase made up of two parts. The first part aimed at returning approaches to finding or detecting performance problems:

```
( "detecting" OR
 "finding" OR
 "profiling" )
AND
( "performance problems" OR
"performance issues" OR
"performance bugs" OR
"performance bottlenecks" OR
"performance antipatterns" OR
"performance anti-patterns" OR
"object churn" OR
"memory bloat" OR
"runtime bloat" )
```

The second part of the search criteria is aimed finding approaches to profiling code, because profiling based approaches are such a ubiquitous method of understanding program performance:

```
( "profiling" OR "profiler" )
AND
( "calling context" OR
"path profiling" OR
"call-path" OR
"call-graph" OR
"call-tree" OR
"calltree" OR
"instrumentation sampling" OR
"sampling based" OR
"instrumentation based" OR
"performance analysis" )
```

We developed the search phrase to be executed against the title, abstract and keywords of the articles in each electronic database. The development of our search phrase was completed using the Scopus database. Scopus has comprehensive searching functionality and indexes all of the venues from the initial manual search, making it convenient to evaluate the effectiveness of our searches by comparison against our initial reference set.

This search returned 1,086 results and contained 38/46 = 82% of our reference set. Ideally we would have preferred to achieve a higher recall than 82 percent. We were only able to improve the recall by either including much more generic search terms, which vastly expanded the number of returned results, or by using artificially specific search terms that were derived from the remaining papers and did not return any other relevant results.

3.5 Venues

To select the electronic venues in which to conduct our search we performed a survey of the search engines used in previous systematic literature reviews and screened them

TABLE 1
Inclusion and Exclusion Criteria

Inclusion Criteria	Exclusion Criteria
• Scientific literature written in English and published in peer-reviewed venues in the year 2000 or later • Literature that describes empirical approaches to analysing and understanding performance applicable to object-oriented software • Papers about finding performance problems, bugs or anti-patterns applicable to object-oriented software • Papers about profiling for *performance understanding* applicable to object-oriented software	• Non-empirical (i.e. model-based) performance engineering techniques • Performance optimisations applied by compilers, virtual machines, middleware or hardware • Profiling for purposes other than performance understanding e.g. – reverse engineering software architecture models – program comprehension – workload characterisation and generation – bug detection e.g. deadlocks, memory leaks – intrusion detection • Performance monitoring approaches not focussed on improvement e.g. – detecting quality of service violations – performance prediction – service provisioning – capacity planning • Approaches specific to domains other than object-oriented software e.g. – embedded systems – real-time systems – massively parallel HPC systems – big data systems such as MapReduce applications

for suitability for our review. Zhang et al. [13] reported the results of a similar survey. From these surveys and experiences reported by Dybå et al. [14] and Petersen [15] the final list of electronic databases we chose to search was:

- ACM digital library
- IEEE Xplore Digital Library
- ScienceDirect
- SpringerLink
- SCOPUS
- ISI Web of Science

The other notable electronic databases that were potential candidates for inclusion that we did not include were:

- Wiley InterScience (now Wiley Online)
- Kluwer Online
- EI Compendex
- Inspec

These were excluded because of their high degree of overlap with one or more of the included search venues.

- Kluwer Online has been merged with Springer and is now indexed through the SpringerLink database.
- EI Compendex and Inspec are indexing services that have a high degree of overlap with the SCOPUS and ISI Web of Science indexing services. For example statistics from the JISC Academic Database Assessment Tool (http://adat.crl.edu/) indicate that 3,194 of the 3,610 titles covered by Compendex and 3,142 of the 4,748 titles covered by Inspec are also covered by SCOPUS.

3.6 Article Screening

To build our list of included articles we screened the automated search results from each of the venues searched and applied the inclusion and exclusion criteria listed in Table 1.

Articles were screened by following an adaptive reading depth approach [11], that is the article title and abstract were read first and if possible the article was included or excluded based on the information therein. Otherwise the full text of the article was retrieved and further sections of the article were read as necessary in order to make the inclusion determination.

All articles were screened by the lead author. To help insure the consistency of the screening process two of the other authors independently tested the inclusion and exclusion criteria on an initial set of 100 papers randomly selected from the automated search results. This initial testing showed a generally high level of consistency on the screening results, with the two other authors agreeing with the lead author on 81 and 86 of the 100 papers respectively. The discrepancies were discussed and used to refine the inclusion criteria used for the full screening.

Duplicate papers returned by the searches were excluded. Many papers are indexed in multiple electronic databases and some appear in multiple venues. We detected duplicate papers by manually comparing any papers where the title and lead author matched. There were some examples of similar, but not identical, papers from the same authors appearing in different venues. In this situation we included both papers. Often systematic reviews avoid this by reporting on the underlying primary studies rather than the papers that have been published. However in a mapping study of this size it was not practical to identify the underlying primary study for each paper, so we report simply on the number of papers we found.

3.6.1 *Inclusion and Exclusion Criteria*

Our inclusion criteria reflect the emphasis of our research questions from Section 3.2, in particular:

- We are interested in empirical rather than model-based performance engineering

- We are not interested in automated performance optimisations applied by compilers, virtual machines, middleware or hardware
- We are focussed on approaches applicable to object-oriented software in general and so excluded approaches specific to specialised domains such as embedded, real-time or massively parallel systems.

We regard each of these excluded areas of research as being distinct from and complementary to the research we are focussed on. We discussed these areas of research and how they relate to our area of interest in Section 2.4.

Note that this exclusion of approaches *specific* to other domains does not exclude approaches from those domains that are more general in nature and therefore still applicable to object-oriented software.

3.7 Completing Article Selection

To complete article selection, after the screening of the automated searches, we undertook a final manual search phase. This consisted of two parts, manual searches of selected venues and *reference snowballing*—manually checking paper references for other relevant literature. We felt this final manual search phase was necessary to mitigate the risk of missing relevant literature due to the only moderate recall of the automated searches. We discuss this further in Section 6.

Before the manual search phase we had just over 200 included articles from 100 different venues, including 66 venues with only one article. We chose to perform manual searches against the top five venues which each had contributed more than 10 articles. No other venue had contributed more than 6. These venues were:

- *OOPSLA*—ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications
- *CGO*—International Symposium on Code Generation and Optimization
- *PLDI*—ACM SIGPLAN Conference on Programming Language Design and Implementation
- *ECOOP*—European Conference on Object-Oriented Programming
- *Euro-Par*—International Euro-Par Conference on Parallel Processing

For each of these venues we completed a manual search on all published proceedings from 2000 to 2013. The CGO conference only began in 2003 but otherwise each of these venues had one published proceedings each year. In total our manual venue search covered 66 published proceedings.

3.8 Search Results

The results of our systematic search process are summarised in Table 2. We ran the same logical query against each electronic database but the syntax used is different for each venue due to their different search engine implementations. We report the exact queries we used to allow them to be easily reproduced. For each search we completed the table lists:

- The exact search phrase that was used
- The date the final search was executed
- The total number of results returned by the search
- The number of results that were selected for inclusion after screening
- The number of included results that were unique to that search

Note that because of the overlap in the results returned by the searches (many papers were returned by more than one search) the total number of included results is not equal to the sum of the included results from the individual searches.

The initial automated searches were completed from May to August 2013 but we refreshed all our automated search results in December 2013 when we completed the final manual phases of our search.

There were some particular challenges that we faced with the SpringerLink search. The SpringerLink database search engine did not support searching against the title, abstract and keywords only. It supported either searching against title only (which returned very few results) or against the full text of the articles, including references. This meant that the SpringerLink search returned 26,677 results. In order to practically screen such a large number of results we reverted to screening them in relevance order as ranked by the SpringerLink search engine and stopping once the rate at which we were finding new articles for inclusion had reduced to the point where we felt finding any further articles was unlikely. In practice we screened the first 2,000 results from the search and then stopped because there had been no new included articles in the last 500 results.

Excluding SpringerLink, we had 2,779 automated search results. Including the 2,000 results we screened from the SpringerLink search we screened 4,779 automated search results and from these included 202 articles. The manual searches covered 66 individual conference proceedings and resulted in 32 additional articles. The reference snowballing involved screening many hundreds of references and resulted in 19 additional articles. The final count was 253 included papers, 202 from the screened automated searches, 32 from the manual search and 19 from reference snowballing.

3.9 Search Results Discussion

We will leave the majority of our analysis of the literature until we discuss the results of the data extraction and the threats to validity for our study but there are some interesting conclusions that can be drawn from the search results we have reported in Table 2.

The first is that each phase of the search (the automated searches, the manual search and the reference snowballing) was a valuable part of the search process. Each led us to discover and include literature that would not otherwise have been covered. That this was true for our mapping study does not necessarily mean it would be true for all systematic reviews. Reviews with a narrower focus may find that either the automated search phase or the manual search phase is superfluous, but that was certainly not our experience.

What was interesting was the very small number of unique search results returned by some of the automated searches. Indeed the ACM, IEEE and ScienceDirect searches together only returned six unique results not returned by either the SCOPUS or Web of Science index searches. The

TABLE 2
Systematic Search Results

Venue	Search Details	Date	Results	Included	Unique
ACM digital library - performance search	((Title:("object churn"OR "memory bloat" OR "runtime bloat" OR "performance bugs" OR "performance antipatterns" OR "performance anti-patterns" OR "performance problems" OR "performance issues" OR "performance bottlenecks") OR Abstract:("object churn" OR "memory bloat" OR "runtime bloat"OR "performance bugs" OR "performance antipatterns"OR "performance anti-patterns" OR "performance problems"OR "performance issues" OR "performance bottlenecks") OR Keywords:("object churn" OR "memory bloat"OR "runtime bloat" OR "performance bugs" OR "performance antipatterns" OR "performance anti-patterns"OR "performance problems" OR "performance issues"OR "performance bottlenecks")) AND (Title:("detecting"OR "finding" OR "profiling" OR "performance understanding") OR Abstract:("detecting" OR "finding"OR "profiling" OR "performance understanding") OR Keywords:("detecting" OR "finding" OR "profiling"OR "performance understanding")))	18 Dec 2013	106	24	0
ACM digital library - profiling search	((Title:("calling context"OR "path profiling" OR "call-path" OR "call-graph"OR "call-tree" OR calltree OR "instrumentation sampling"OR "sampling based" OR "instrumentation based"OR "performance analysis") OR Abstract:("calling context"OR "path profiling" OR "call-path" OR "call-graph"OR "call-tree" OR calltree OR "instrumentation sampling"OR "sampling based" OR "instrumentation based"OR "performance analysis") OR Keywords:("calling context"OR "path profiling" OR "call-path" OR "call-graph"OR "call-tree" OR calltree OR "instrumentation sampling"OR "sampling based" OR "instrumentation based"OR "performance analysis")) AND (Title:(profiling OR profiler) OR Abstract:(profiling OR profiler) OR Keywords:(profiling OR profiler)))	18 Dec 2013	150	46	0
IEEE Xplore - performance search	(("object churn" OR "memory bloat" OR "runtime bloat" OR "performance bugs"OR "performance antipatterns" OR "performance anti-patterns"OR "performance problems" OR "performance issues"OR "performance bottlenecks") AND ("detecting"OR "finding" OR "profiling" OR "performance understanding"))	18 Dec 2013	105	12	1
IEEE Xplore - profiling search	((("calling context" OR "path profiling" OR "call-path" OR "call-graph"OR "call-tree" OR calltree OR "instrumentation sampling"OR "sampling based" OR "instrumentation based"OR "performance analysis") AND ("profiling"OR "profiler")))	18 Dec 2013	297	38	4
ScienceDirect	TITLE-ABSTR-KEY((("object churn" OR "memory bloat" OR "runtime bloat" OR "performance bugs"OR "performance antipatterns" OR "performance anti-patterns"OR "performance problems" OR "performance issues"OR "performance bottlenecks") AND ("detecting"OR "finding" OR "profiling" OR "performance understanding")) OR (("calling context" OR "path profiling" OR "call-path" OR "call-graph"OR "call-tree" OR calltree OR "instrumentation sampling"OR "sampling based" OR "instrumentation based"OR "performance analysis") AND (profiling OR profiler)))	18 Dec 2013	80	9	1
SpringerLink	(("object churn" OR "memory bloat"OR "runtime bloat" OR "performance bugs" OR "performance antipatterns" OR "performance anti-patterns"OR "performance problems" OR "performance issues"OR "performance bottlenecks") AND ("detecting"OR "finding" OR "profiling" OR "performance understanding")) OR (("calling context" OR "path profiling" OR "call-path" OR "call-graph"OR "call-tree" OR calltree OR "instrumentation sampling"OR "sampling based" OR "instrumentation based"OR "performance analysis") AND (profiling OR profiler))	18 Dec 2013	26677[a]	61	40
Scopus	TITLE-ABS-KEY((("object churn" OR "memory bloat"OR "runtime bloat" OR "performance bugs" OR "performance antipatterns" OR "performance anti-patterns"OR "performance problems" OR "performance issues"OR "performance bottlenecks") AND ("detecting"OR "finding" OR "profiling" OR "performance understanding")) OR (("calling context" OR "path profiling" OR "call-path" OR "call-graph"OR "call-tree" OR calltree OR "instrumentation sampling"OR "sampling based" OR "instrumentation based"OR "performance analysis") AND (profiling OR profiler)))	18 Dec 2013	1086	137	16
Web of Science	TS=((("object churn" OR "memory bloat"OR "runtime bloat" OR "performance bugs" OR "performance antipatterns" OR "performance anti-patterns"OR "performance problems" OR "performance issues"OR "performance bottlenecks") AND ("detecting"OR "finding" OR "profiling" OR "performance understanding")) OR (("calling context" OR "path profiling" OR "call-path" OR "call-graph"OR "call-tree" OR calltree OR "instrumentation sampling"OR "sampling based" OR "instrumentation based"OR "performance analysis") AND (profiling OR profiler)))	19 Dec 2013	955	102	7
Manual Search	Euro-Par 2000-2013, OOPSLA 2000-2013, PLDI 2000-2013, CGO 2003-2013, ECOOP 2000-2013	Dec 2013		98	32
Reference Snowballing		Dec 2013			19
Total (duplicates removed)			**2779**[b]	**253**	

[a] *See discussion on SpringerLink in section 3.8*
[b] *Excluding SpringerLink and Manual Searches*

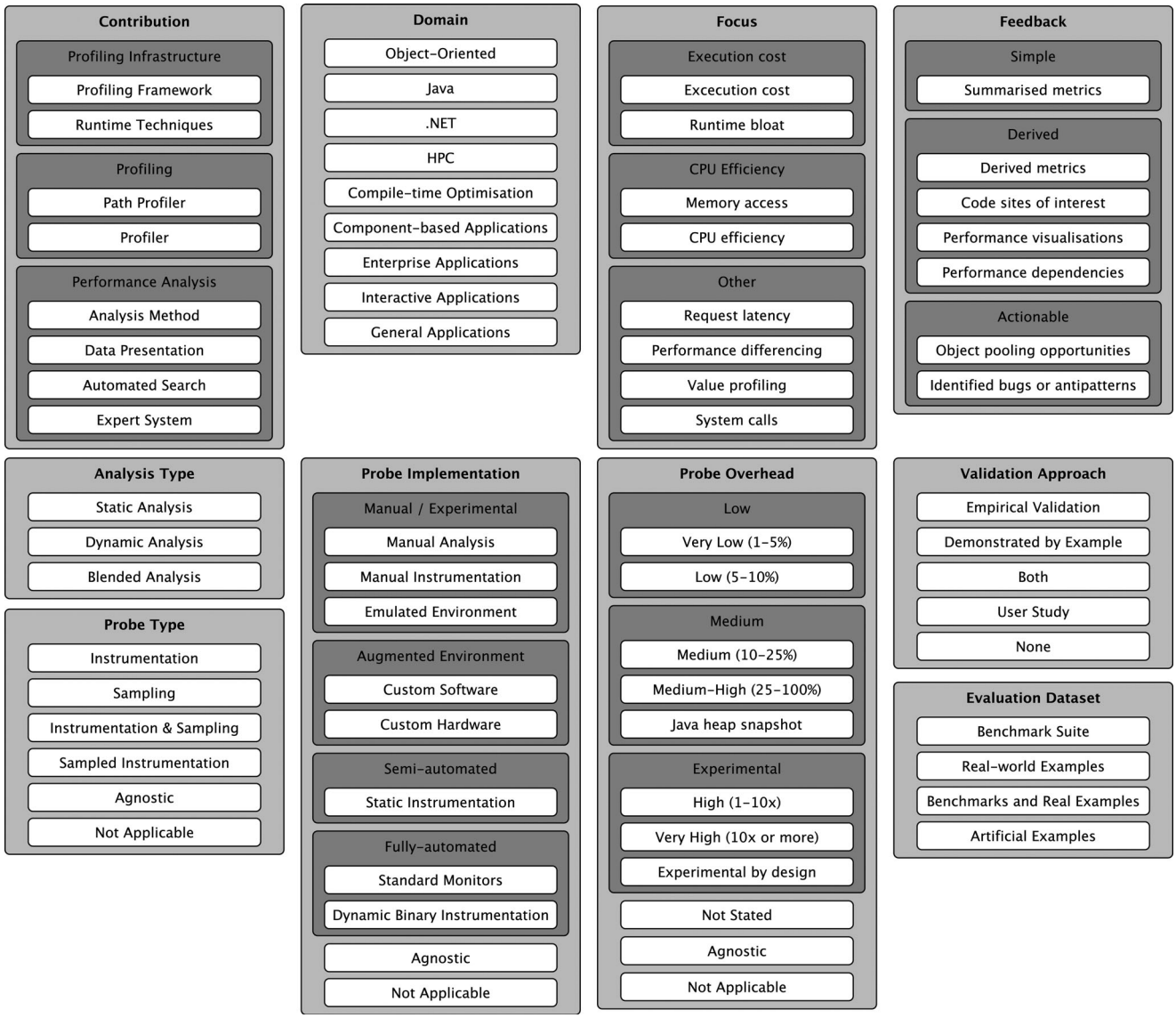

Fig. 1. Facets and categories.

implication of this is that the manual search and reference snowballing phases were more productive and more critical to the coverage of our literature review than the automated searches against those databases.

3.10 Data Extraction

Data extraction was completed by reviewing each included article and categorising it according to a classification scheme that we developed to allow us to structure and map the literature in our review. We used an adaptive reading depth approach, similar to our approach when screening articles for inclusion, beginning with reading the title, abstract and introduction and then reading further sections as required to complete our classification.

3.11 Classification Scheme

Our classification scheme allows us to structure the literature in our study to map the literature in general and answer our research questions in particular. The classification scheme we have developed is a novel contribution in its own right, providing a framework for categorising and describing performance engineering approaches applicable to object-oriented software. Our scheme consists of 10 facets, shown in Fig. 1.

Our facets were selected to capture both attributes that are useful to help us answer our research questions (such as Feedback and Probe Implementation) and attributes that represent commonly recognised aspects of performance analysis approaches (e.g. Analysis Type and Probe Type). We did not predetermine the categories for each facet ahead of time, instead using a *keywording* approach similar to that described by Petersen et al. [11]. We classified each paper's approach using key words or phrases for each facet that describe the characteristics of the approach. We then consolidated the list of key phrases that had been used for each facet into a coherent set of categories, generalising sets of phrases into a single category where necessary.

3.11.1 Contribution

The contribution of a paper is the type of novel contribution it is providing to the field of research. Capturing the contribution allows us to compare papers that broadly had the

same objectives. Our contribution facet contained 8 categories in total that were grouped into 3 broader categories:

- Profiling infrastructure papers describe a general purpose approaches relevant to the collection of performance data e.g. binary instrumentation techniques
- Profiling papers are focussed on a technique for collecting a specific type of performance data e.g. path, edge or calling context profilers
- The analysis approaches are focussed on how performance data is processed or analysed. This includes techniques for graphing, navigating and searching performance data as well automated search and pattern matching approaches that seek to highlight interesting information in the performance data.

3.11.2 Domain

The domain is the particular research domain that motivated the approach. Mostly this is the type of software that the approach is applied to e.g. object-oriented software or massively parallel HPC applications, but there are also approaches specific to a particular technology framework such as Java or .NET (that are not generalisable to all object-oriented software). Having this facet allows us to contrast and compare results from different research domains or communities.

3.11.3 Focus

The focus of an approach is the particular type of performance information that the approach investigates. Performance of software is a wide-ranging concern and this facet allows us to classify more specifically the types of performance that the literature was interested in. Approaches that are focussed on the use of runtime resources (using measures such as CPU time or invocation counts) we classify as execution cost approaches. The other major group of approaches are those that focus on understanding and improving the efficiency of CPU execution, either by specifically tracking the instructions per cycle rate for an execution or by monitoring the rate of events that reduce efficiency, such as memory accesses, memory cache misses or branch mispredictions. Additionally there are a number of other more specialised focii such as request latency profiling or profiling the use and cost of system calls. Request latency profiling is investigating the time taken for an application to respond to an incoming request (such as an HTTP request or a UI trigger event) and is important to improve the responsiveness of interactive applications.

3.11.4 Feedback

The feedback provided by an approach is the style of information or data generated for the user. This is one of the primary motivations for our study, to investigate the style and richness of the feedback that is generated for the user about the performance of software. The simplest feedback returns straight-forward raw or summarised metrics, essentially a long list of measurements often collated by instruction, method or calling context (the hierarchy of active methods calls leading to the current call). The derived feedback approaches perform some level of calculation or interpretation of results. This includes those that generate an advanced visualisation of the performance data (over and above a simple graphing of the summarised metrics), those that calculate new derived metrics from the raw measurements and those that are able to highlight specific code sites of interest. The approaches we regard as providing actionable feedback are those that are able to identify concrete opportunities where a known solution can be applied to improve performance. This includes the identification of object pooling opportunities and finding known coding bugs or anti-patterns.

3.11.5 Analysis Type

Analysis type is the style of analysis used by the approach i.e. whether the approach used static analysis, dynamic analysis or blended (both static and dynamic) analysis. Static analysis approaches are those that rely purely on an analysis of the source code or statically generated artifacts and do not observe the software during its execution. Dynamic analysis approaches are those that rely purely on measurements and analysis of data gathered from software that is executing. Blended analysis, described as a paradigm by Dufour et al. in 2007 [16], uses both static and dynamic analysis information during their application. There are approaches that use basic static information (for example basic block or method information) to insert instrumentation to collect dynamic measurements. Given the simple nature of this static information we still regard these as pure dynamic analysis approaches.

3.11.6 Probe Type

This facet and the two following all capture some aspect of the mechanism used to record dynamic measurements. As such they are not applicable to pure static analysis approaches (and hence each have a not applicable category). Additionally they each have an agnostic category to categorise performance analysis approaches that are applied to data after it is captured and are independent of how the data was captured.

Probe type describes the type of mechanism used to capture dynamic measurements. Broadly speaking these are either sampling based or instrumentation based. Sampling based approaches observe the state of the system at sampled points in time and build a statistical profile of the dynamic behaviour based upon these samples e.g. java stack sampling. Instrumentation based approaches use probes inserted at specific points in the system to record exact measurements every time the probe is activated e.g. recording every method invocation. Some approaches use sampled instrumentation, this is when instrumented probes are used to record measurements but only during certain sampling periods i.e. they are not permanently active.

3.11.7 Probe Implementation

The probe implementation is the actual mechanism used to capture dynamic measurements, of which there are a wide variety. For example the application source code may be manually modified to include code for logging specific measurements (manual instrumentation), modifications

might be made to a kernel library to record how often it is invoked (custom software), or there may be standard APIs that can be used to retrieve standard measures (standard monitors).

Capturing the probe implementation allows us to investigate which mechanisms are popular and gives some insight into how challenging to apply the approach is in practice, as some mechanisms are more accessible than others. We can group the different categories we have for probe implementation into:

- those that can only be applied manually or in an experimental fashion
- those that require an augmented environment with either customised hardware or patched system software, standard libraries or middleware
- static instrumentation approaches that require an extra build-time step but are otherwise automated
- fully automated approaches based on standard monitors or dynamic binary instrumentation that can be activated dynamically with already running applications

3.11.8 Probe Overhead

The probe overhead is the amount of execution cost overhead imposed by the dynamic measurement capture. Capturing overhead allows us to classify the impact of the approaches to judge how practical they are to apply.

3.11.9 Validation Approach

This facet and the next are concerned with how the approaches proposed in the literature are evaluated.

The validation approach is the type of evidence that is put forward to establish the effectiveness of a technique. This can include empirical evaluation, where some measurable aspect of the technique (such as its overhead or accuracy) is analysed to prove its effectiveness, demonstration by example, where a case study gives a detailed report on using the technique to achieve a useful outcome, or a user study, where human subjects use the technique and report on its utility.

3.11.10 Evaluation Dataset

The evaluation dataset for an approach is the type of input used for the described evaluation. Many evaluations are completed using a standard suite of benchmarks (often specific to the particular domain of the research). Others use real-world applications as input, either large open-source frameworks or real-world industrial applications with realistic attributes. Some approaches are evaluated over artificial examples that have been constructed specifically for the purpose of demonstrating or evaluating the approach.

4 Results

4.1 Overview

In total, we included 253 papers in our completed review. With ten different facets and a number of other pieces of data (such as the publication year and publication venue) defined for each paper there is a large amount of data available for analysis. The complete list of included papers is given in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2015.2396514, and the complete categorisation for all 253 included papers is given in Appendix B, available in the online supplemental material. Here we focus on presenting the results that address our specific research questions.

We begin by giving an overview of the included literature in addressing research question 1. The categorisation results for the facets shown in Figs. 3 and 8 are discussed in answering the other research questions.

4.2 Research Question 1

What approaches to empirical performance analysis have been proposed that are applicable to object-oriented software?

This narrative summary and Fig. 2 describes the literature using selected facets in a way that aims to provide a useful overview of the literature. For this purpose we have primarily used the results from the contribution and focus facets as these best capture the particular problem or area the paper was addressing. Note that in order to improve readability Fig. 2 is not drawn precisely to scale, but the relative sizes of the labelled areas are approximately in proportion to the number of papers they represent.

We have first divided the papers into three groups representing the type of novel contribution they provide:

- 138 Profiling papers
- 77 Performance Analysis papers
- 38 Profiling Infrastructure papers

The profiling papers are focussed on a technique for collecting a specific type of performance data. The performance analysis papers describe an approach to analysing or interpreting the performance data. In most cases the distinction is clear, indeed many profilers essentially do no data analysis and many analysis approaches are completely agnostic of how the data is collected, but in some cases where the approach includes both the collection and analysis of data the distinction can be a fine one.

The profiling infrastructure papers are those that describe tools and runtime techniques that are useful when creating profilers or other performance understanding tools but do not in-of-themselves describe a profiling or performance understanding approach. These papers are still applicable to object-oriented software performance, and therefore included in our review, because they directly support creating new or improving existing performance understanding approaches. There are 18 runtime technique papers. These include the probabilistic [s40][1], inferred [s161] and precise [s206], [s207] calling context encoding approaches, which describe low overhead approaches to establishing the current full calling context, and papers on instrumentation approaches [s35], [s72], [s160] or aspect oriented programming approaches [s8], [s32], [s222] aimed at supporting dynamic analysis in general and profiling in particular. 20 papers describe tools and frameworks designed to support dynamic analysis in general and profiling in particular. Some of these are instrumentation tools

1. Paper references beginning with s refer to the List of Included Papers in Appendix A, available in the online supplemental material.

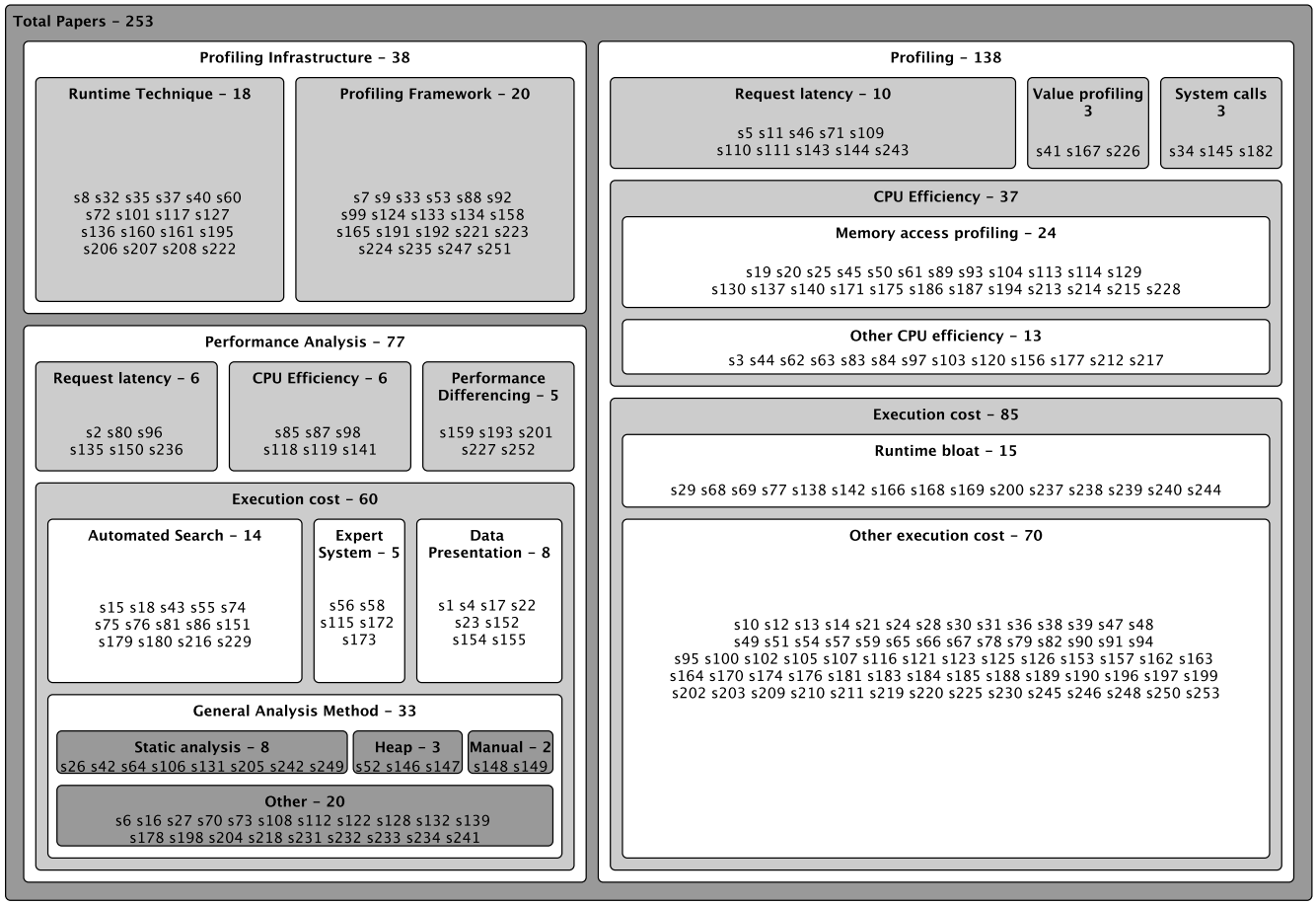

Fig. 2. Literature overview by contribution and focus.

such as Pin [s133] or Valgrind [s165] and others are tools which facilitate the parallelisation of dynamic analysis [s92], [s158], [s224], [s247], [s251] to reduce its cost.

There are 138 papers that describe a form of profiler. Of these there are a small number that have a highly specialised focus, such as the three papers concerned with measuring the amount of time spent in kernel code or making operating system calls [s34], [s145], [s182] and the 3 papers concerned with value profiling—tracking the most commonly returned values from individual functions or statements [s41], [s167], [s226]. Ten papers are focussed on understanding request latency rather than overall application performance or throughput. These are either specifically looking at unresponsive UIs in interactive desktop applications [s71], [s109], [s110], [s111], or investigating the reasons for slow responses from multi-tier web-based enterprise applications [s5], [s11], [s46], [s143], [s144], [s243]. The 37 CPU efficiency papers are focussed on monitoring and understanding the code that cause a CPU to run at less than maximum performance. All of these leveraged hardware performance monitors to track low level events that cause CPU pipeline stalls and a drop in overall computation speed. The majority of these (24 papers e.g. [s129], [s171], [s175], [s213], [s214],) are specifically interested in profiling memory accesses to understand where and why memory loads, stores and cache misses are occurring as these are often the main culprits in stalling the CPU. The remaining 85 profiling papers focus on execution cost generally, that is understanding which code regions are consuming the most computation resources. A small subset of these (15 papers e.g. [s68], [s69], [s166], [s168], [s169], [s237], [s238], [s239], [s240], [s244]) are interested in runtime bloat in object-oriented software.

There are 77 papers concerned with analysis of performance data. As with the profiling approaches we can classify these by focus and again we find that there are a small number of papers with a specialised focus. Five papers look at performance differencing approaches [s159], [s193], [s201], [s227], [s252], that aim to highlight changes in performance between two captured profiles. There are 6 request latency papers [s153], [s193], [s201], [s227], [s252] with the same goal as the profiling approaches that focussed on request latency—to understand the reasons for unresponsive UIs or slow responses from web-applications. There are only six analysis papers that look to understand CPU efficiency [s85], [s87], [s98], [s118], [s119], [s141], a much lower ratio than for the profiling papers, and the remaining 60 papers focus on execution cost generally. The 14 automated search papers describe approaches where the performance analysis system automatically searches for code regions that exhibit certain pre-defined performance characteristics. Most are systems similar in approach to Periscope [s18], [s81], [s85], [s86] and Paradyn [s43], [s55], [s179], [s180] which run a series of systematic experiments, monitoring progressively finer grained regions of code and measuring and evaluating the performance metrics of interest to narrow down the regions of

TABLE 3
Categorisation by Venue

Venue	Description	Count
Euro-Par	International Euro-Par Conference on Parallel Processing	26
OOPSLA	Conference on Object-Oriented Programing, Systems, Languages, and Applications	22
CGO	International Symposium on Code Generation and Optimization	21
PLDI	Conference on Programming Language Design and Implementation	18
ECOOP	European Conference on Object-Oriented Programming	11
PPPJ	International Symposium on Principles and Practice of Programming in Java	6
ICSE	International Conference on Software Engineering	6
Tools for HPC	International Workshop on Parallel Tools for High Performance Computing	5
SPE	Software: Practice and Experience	5
ISPASS	International Symposium on Performance Analysis of Systems and Software	5
ICPE/WOSP	International Conference on Performance Engineering	5
SIGMETRICS	International Conference on Measurement and Modeling of Computer Systems	4
SC	Supercomputing Conference	4
CCPE	Concurrency and Computation: Practice and Experience	4
5 more venues with 3 papers		
15 more venues with 2 papers		
66 more venues with 1 paper		

interest. Five papers [s56], [s58], [s115], [s172], [s173] are approaches that employ an expert system, rules based, approach to analysing a given set of performance measurements and identifying known patterns of undesirable behaviour. There are 8 papers that propose new approaches to presenting performance data in order to aid the interpretation or comprehension of that data. Most of these relate to visualising or presenting the hierarchical data contained in large calling context trees, for example using calling context ring charts [s152], [s154], [s155]. Execution profiling blueprints [s22], [s23] is an approach to displaying multiple performance metrics or characteristics in the one visualisation to aid in drawing performance inferences.

The remaining 33 papers are general analysis papers. 8 of these are static analysis approaches to understanding performance. Several of these static analysis papers are a type of feedback generated from static optimisers [s64], [s205] (normally but not necessarily associated with compilers) that indicate to the user where code changes could be made to enable further optimisations. Others statically estimate loop, path or method execution frequency [s42], [s131], [s249]. One approach looks for code patterns that equate to known performance bugs [s106]. The patterns have been derived from an in-depth study of performance bugs in open source bug databases. 3 of the other general analysis approaches are based on an analysis of Java heaps to understand memory performance [s52], [s146], [s147], generally looking for patterns that indicate wasteful or inefficient data structures. There are two general analysis papers that rely on customised manual experiments to investigate the way complex framework-based applications transform data [s148], [s149], to understand the reasons for inefficient computation. The remaining 20 papers are a collection of different approaches that focus on the analysis of dynamic performance data to understand the execution cost of applications. These include techniques for aggregating or summarising performance across user-defined code regions [s128], [s204], [s231], [s232], or using user-defined metrics [s73], [s108], [s198].

4.3 Research Question 2

How can these approaches be characterised?

We have already given an overall description of the literature in our review in the previous section. Here we describe some selected results and trends that are apparent in our classification that are not covered in more detail elsewhere.

4.3.1 Venue

One of the traditional results to report in a mapping study is a breakdown of the literature by the publication venue (conference or journal) as this provides an indication for researchers as to which venues are likely to have the most interest in similar research. Our breakdown, shown in Table 3, highlights a number of interesting things:

- Literature related to performance has been published in a very broad range of venues, our survey includes literature from 100 different publication venues, including 66 venues that only provided a single paper.
- Although the performance focussed venues such as the International Conference on Performance Engineering and International Symposium on Performance Analysis of Systems and Software (ISPASS) did contribute a number of articles (five from each venue) the most important venues for the empirical approaches to performance analysis that we were interested in were the programming and languages focussed conferences such as OOPSLA, PLDI and ECOOP (which contributed over 50 articles between them).[2]

4.3.2 Domain

The categorisation of the articles by research domain (shown in Fig. 3) again gives some insight into the breadth of the research that has been conducted into software performance. Despite our inclusion criteria restricting us only to articles applicable to object-oriented software and excluding approaches specific to other domains we still had a large contribution from the HPC and compile-time optimisation domains. We made a distinction between articles that were

2. Excluded ICPE papers focussed on areas, such as model-based performance prediction, which are related to but not covered by our survey. We discussed these areas in Section 2.4.

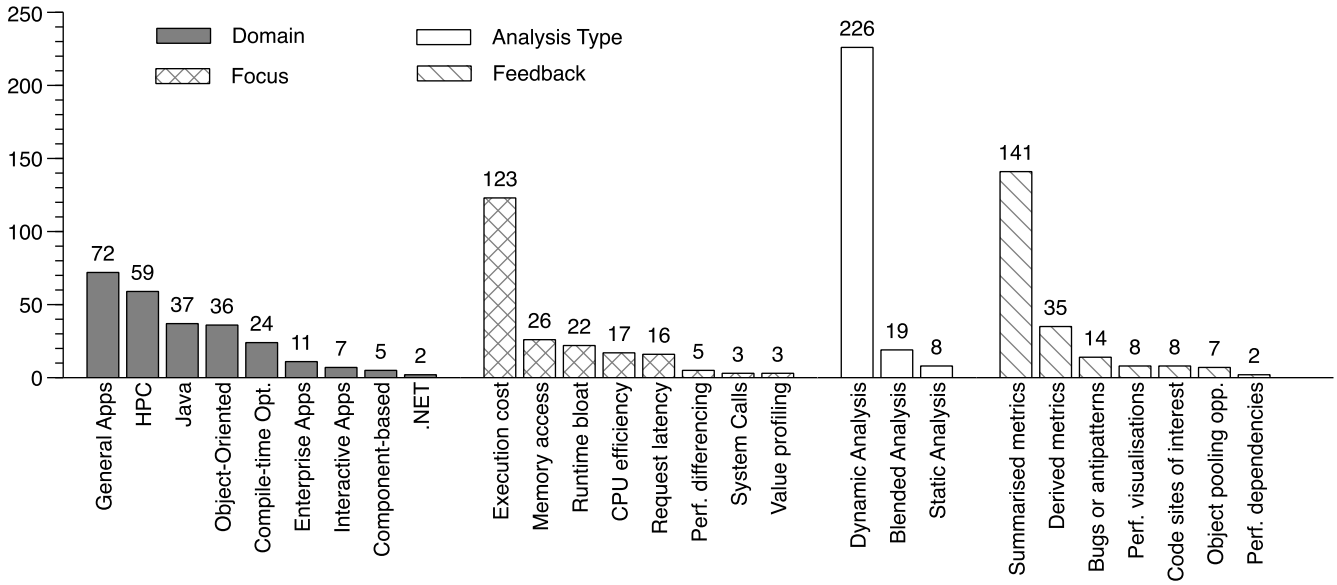

Fig. 3. Categorisation results—Domain, focus, analysis type and feedback

focussed on object-oriented applications in general and those that focussed on applications built with the Java or .NET frameworks. Of course Java and .NET are object-oriented language frameworks, but some of the approaches were focussed not on the object-oriented nature of these frameworks but something specific to the framework design, such as techniques for efficient profiling in the JVM. One of the interesting things we noticed here was the great contrast between the number of papers relating specifically to Java (37) compared to .NET (2). Given the importance of .NET as a commercial development platform it would seem to be severely underrepresented in the literature.

4.3.3 Focus

The Focus categorisation results shown in Fig. 3 include only the 215 profiling and performance analysis papers. We excluded the 38 profiling infrastructure papers as they were all general purpose approaches applicable when investigating a number of different performance concerns i.e. they had no specific focus.

The most obvious trend here is that by far the majority of the literature (66 percent—142 out of 215 once the runtime bloat literature is included) is focussed on analysing the execution cost of applications. Perhaps the surprising thing though is not that this number is so high but that it is so low, that there are fully a third of the papers that investigate other aspects of performance besides the raw time and resources they consume. In particular 20 percent (43 out of 215) of the literature focuses on CPU efficiency and memory access profiling, marking it as a significant form of performance analysis that attracts little attention in some domains. A bubble plot of Focus versus Domain (see Fig. 5) shows the relative lack of interest in CPU efficiency from the Object-Oriented and Java domains.

There are some other interesting points illustrated in the bubble plot. Unsurprisingly, given its nature, we can see that the concept of runtime bloat is one that mostly exists in the Object-Oriented domain. Finally we can see the particular focus on request latency profiling for Enterprise and Interactive applications.

There are also some noticeable trends over time for the Focus facet, as shown in Fig. 4. Whilst the amount of activity in some focus categories remains relatively steady over time, such as execution cost, memory access profiling and CPU efficiency, there is a clear upward trend for both request latency profiling and runtime bloat.

4.3.4 Analysis Type

The clear trend here (Fig. 3) is the dominance of pure dynamic analysis approaches. The lack of pure static analysis approaches is not surprising, given that performance is a runtime property. However the small number of blended analysis approaches indicates there are likely opportunities to leverage more static analysis in the performance analysis process.

When we look at the trends over time for analysis type (Fig. 6) we can see that both blended and static analysis have become more common in recent years.

4.4 Research Question 3

What form of feedback does the approach provide?

The focus of this research question is to try to understand how helpful the approaches are to the end goal of

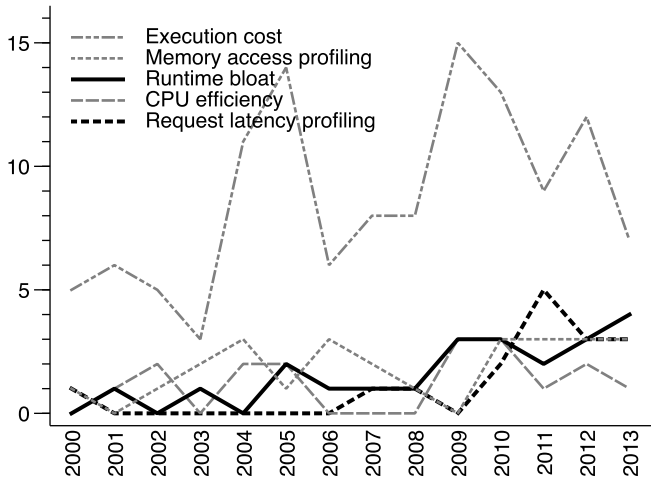

Fig. 4. Focus over time.

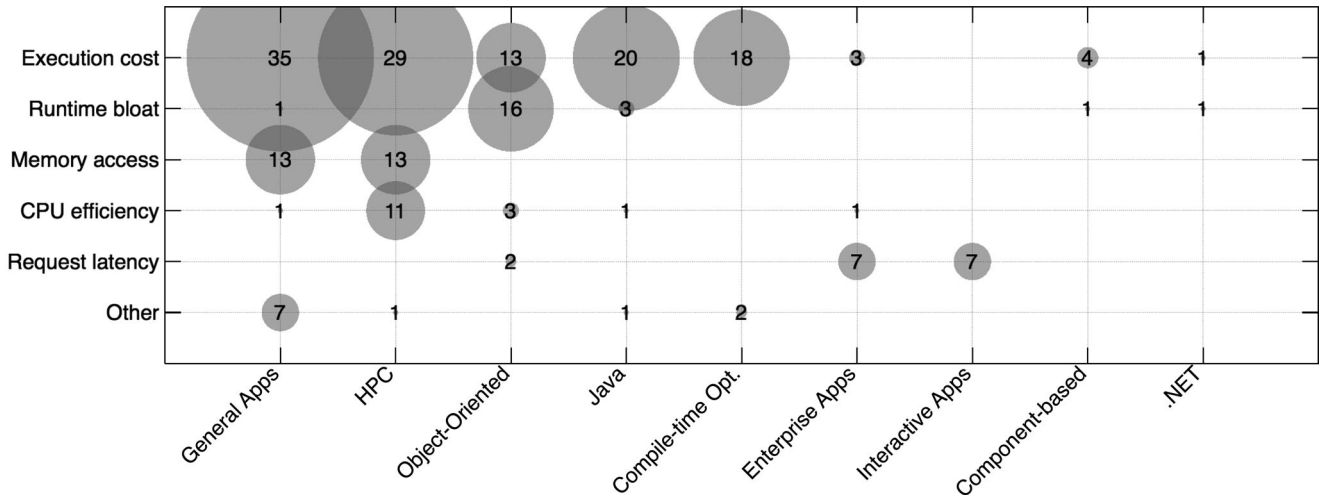

Fig. 5. Focus versus domain.

improving software performance. We captured the type of feedback provided by each approach in the Feedback Type facet (Fig. 3). As with the earlier Focus facet we excluded the 38 profiling infrastructure papers from this categorisation as they did not describe approaches that generated performance understanding feedback.

Of the remaining 215 papers, 141 of them provide only raw or summarised metrics. These are metrics that may be helpful in understanding simple properties of an application, such as the absolute cost of a particular method or execution path, but generally require a significant amount of interpretation to find useful performance improvements. In particular they tend to only indicate where an application is consuming a large number of resources but not whether this consumption is wasteful or useful or how to reduce the consumption.

There are 53 approaches that generate some form of derived feedback by performing additional analysis or transformation on the raw performance data. These include:

- 35 derived metrics
- Eight approaches to advanced visualisation or presentation of performance information expressly designed to aid its interpretation
- Eight code sites of interest
- Two performance dependencies

In general the derived feedback approaches aim to make the interpretation of the performance data easier. They generally create or define a higher-level abstraction for which they are able to provide a metric that not only helps indicate where an application is consuming resources but why this may be occurring or helps imply a potential solution. Examples include approaches for finding inefficiently used data structures, automated search approaches that search for execution paths that exhibit particular undesirable behaviours and various other novel performance metrics.

Finally there are 21 approaches that generate concrete actionable feedback. Seven of these approaches identify concrete locations where object pooling was both possible and desirable. These use a variety of object lifetime and object usage profiling techniques to establish which allocation sites create objects that are suitable for being pooled and reused. The other 14 actionable feedback approaches identify known performance bugs or antipatterns in concrete source code locations. These cover a wide range of different types of performance problems such as inefficient memory access patterns, inappropriate data structures and never-used or rarely-used object allocations. Several of the antipattern identification approaches make use of configurable detection rules allowing expert users to define new antipatterns.

Overall the majority of approaches (141/215) still only provide simple feedback.

There are also some interesting trends over time (Fig. 7). The obvious trend is the increase in approaches looking at more sophisticated feedback in recent years, object pooling opportunities and identified bugs or antipatterns in particular.

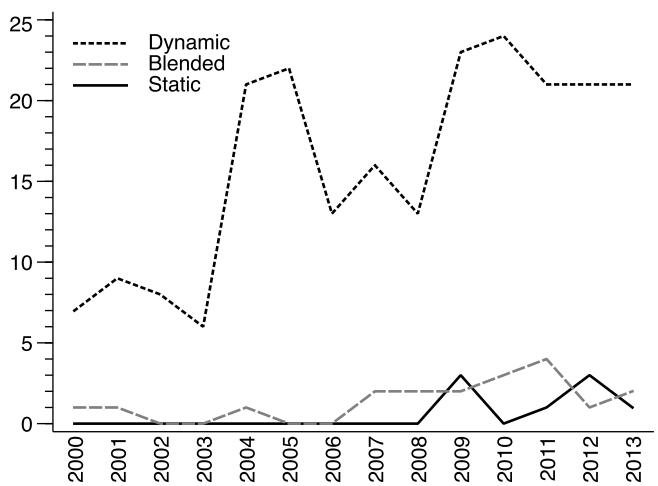

Fig. 6. Analysis type over time.

4.5 Research Question 4

How practical to apply are the approaches?

(1) *What is required to apply the approach?*
(2) *Can the approaches be applied to live production environments?*

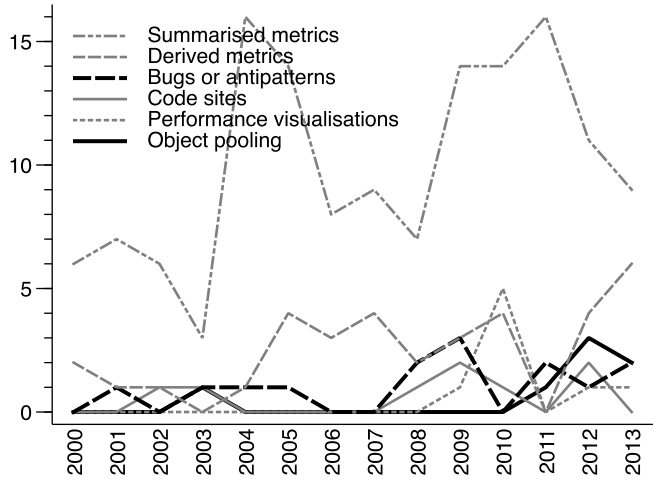

Fig. 7. Feedback over time.

This research question was motivated by the interest practitioners would have in the practicality of applying any performance monitoring or analysis approach. Approaches that are complex to apply can still be useful; there is opportunity within the development life-cycle of many projects to use time-consuming approaches if they provide valuable information. However it is advantageous if approaches can be simply and rapidly applied because it means they can be used more often and in more situations.

We had two facets that were primarily aimed at helping us answer this question, Probe Implementation and Probe Overhead. The categorisation results for these facets are shown in Fig. 8. They exclude the eight pure static analysis approaches, which do not use dynamic probes and therefore do not have an implementation or impose any overhead. It is worth noting that in most situations the static analysis approaches are the most practical to apply because they do not require a running environment. This is one of their great advantages and means that they can be fully integrated into development environments and processes with relative ease. However there are only a few such approaches in the literature we reviewed.

The results of the Probe Implementation categorisation indicate that there are a significant number of approaches (39 percent—96 out of 245) that rely on either dynamic binary instrumentation or standard monitors. These are approaches that we regard as being relatively straight-forward to apply as they do not require a customised environment or build-time instrumentation, the approach can be used in any standard environment with pre-existing binaries. There are a further 34 approaches (~14 percent) that are analysis approaches that can work with data obtained from any suitable performance monitor and therefore should also be practical to use. The other approaches mostly rely either on custom hardware or software, or require static or manual instrumentation of the application before they can be used. This doesn't preclude them being used in real-world environments but does require extra effort to facilitate their use.

The Probe Overhead categorisation is more difficult to interpret for several reasons. The high number of papers (58 out of 245) that don't address the overhead of their dynamic analysis approach introduces significant uncertainty about any conclusions that are drawn. It also is not clear precisely what the cut-off point is for 'acceptable' levels of overhead, generally this will depend on the requirements of the system being analysed. Another complication is that interpreting some of the reported overheads for the different approaches is difficult due to the variety of approaches taken to measuring and reporting overhead. Whilst many papers take a rigorous approach involving assessing overhead against a variety of standard benchmark and realistic real-world applications others simply used a single case study or report only comparative overhead results relative to other profiling techniques, rather than a comparison with the unprofiled application. Also many of the approaches (especially those using execution sampling methods) could be tuned to reduce overhead at the cost of accuracy. In these cases

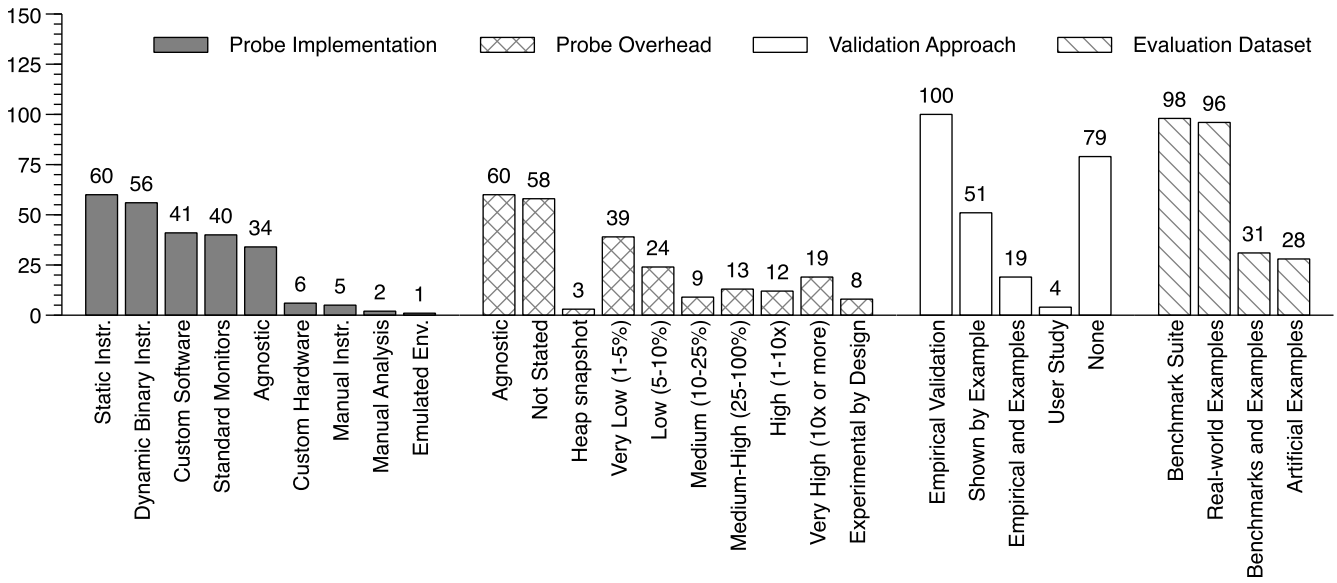

Fig. 8. Categorisation results—Implementation and evaluation.

the literature often suggested a particular overhead/accuracy point as the best-case scenario and those are the figures we used for our categorisation. We have categorised 60 papers as *agnostic* where the overhead is either not measurable or not relevant. These are either profiling infrastructure papers where it is impossible to assess overhead without knowing the actual analysis implemented using the infrastructure, or analysis approaches that focus on the post-processing of previously collected data and so the approach itself is independent of the style of data collection.

What is clear from the results is that we do have a significant number of approaches, about 25 percent, that have an emphasis on minimising overhead, those with Very Low or Low reported overhead. The majority of the remaining approaches either have a high overhead or do not report on overhead at all, there are relatively few approaches that fall into the medium or medium-high overhead areas.

When we look at the categorisation of approaches by both implementation approach and overhead we see that a significant proportion of the low overhead approaches are achieved by using custom hardware or software implementations. Overall only around 10 percent (25 out of 245) of the approaches are both low overhead and fully automated.

The other trend that is apparent in our data is the type of feedback generated for low overhead approaches is dominated by simple summarised metrics. Nearly 90 percent of low overhead approaches return only summarised metrics as opposed to around 65 percent of all profiling and analysis method approaches.

4.6 Research Question 5

How are approaches evaluated and validated?

We were interested in how approaches to improving performance engineering were evaluated and validated. This followed from our interest in the utility of performance analysis approaches, to understand how effective reported performance analysis approaches were we wanted to understand how they had been evaluated.

We used two facets to classify how approaches were evaluated, the Evaluation Dataset and the Validation Approach. The categorisation for these facets is shown in Fig. 8. There are two notable findings from the Validation Approach classification:

1) A significant proportion of the approaches ($79/253 \sim 31$ percent) did no validation of their approach.
2) Only a tiny minority ($4/253 \sim 1.6$ percent) validated their approach with real users in a user study.

The majority of the approaches that were validated were done so via either empirical validation or a case study that demonstrated using the approach to achieve a desirable outcome. The empirical validations were generally quantitative comparisons of accuracy or overhead compared to some established baseline. In some cases, where it was possible to have an oracle that could identify true positives, the empirical validation included measuring false positive and false negative rates generated by the tool. Most of the 79 papers that we have classified as doing no validation did demonstrate applying their approach to real-world or benchmark applications but did not validate the output in any way to demonstrate the accuracy or utility of the output. They relied solely on the description of their approach to establish that it was useful. This is in contrast to the 51 papers that demonstrated using their approach with a case study that highlighted how using the approach resulted in information that led to a measurable performance improvement.

The Evaluation Dataset facet classifies the approaches by the type of input they were applied to. Here we see an fairly equal split between using real-world examples (usually large open source or industrial applications) or a standard benchmark suite. There were some approaches that were evaluated over both standard benchmarks and real-world examples. The approaches that did not use either real-world examples or benchmarks used artificial examples. Note that this classification includes all 253 papers, including the 79 that we classified as having not completed any validation. This is because all 253 papers used illustrative examples, even if they were only artificial examples, either when describing or evaluating their approach.

5 Discussion

The intent of our survey was to map and analyse the existing literature pertaining to empirical performance analysis approaches for object oriented software, with a particular emphasis on reviewing the type of performance feedback provided by these approaches. In the previous sections we have described the results of our categorisation of the relevant literature, here we highlight some of those results and discuss some key conclusions we have drawn from our analysis. In particular we highlight where we think there are opportunities for future research.

5.1 Very Little Static or Blended Analysis, Virtually All Dynamic

The analysis type facet gave us the most skewed distribution of results in our entire categorisation. It is clear that not only do pure dynamic analysis approaches dominate in our study but that much of the work done using static or blended analysis (20 out of 27 papers) has been published since 2009. This trend in recent times towards using more static analysis, in conjunction with dynamic analysis or alone, indicates to us that this is an emerging area of work where there are very likely still substantial opportunities. Our definition of blended analysis for performance analysis is not novel, it is specifically described by Dufour et al. [16], but the results of our survey have allowed us to quantify just how infrequently it has been leveraged.

5.2 Cross-Domain Approaches

We found our study covered literature from a very broad range of venues and domains, as we quantified with the categorisation results we covered in Section 4.3. We found this interesting because the focus of our study was literature applicable to object-oriented software and yet we found literature from a wide range of areas that was applicable to object-oriented software. The implication of this is that researchers or practitioners working with object-oriented software need to be aware that there is potentially related

work being done within these other domains, such as HPC or compile-time optimisation. We feel the reverse is likely to be true as well. For example object-oriented software frameworks and concepts are being adopted in some HPC applications and this makes it likely that some of the performance characteristics of object-oriented software will be inherited as well, therefore making the research into object-oriented application performance relevant to the HPC community.

We also found that there were a number of techniques or approaches that had been investigated by researchers from different domains that were similar in nature and therefore likely had some relevance to each other. For example there were expert system approaches to performance classification described by researchers in both the HPC and EJB application domains. Request latency profiling was of interest to researchers from both the enterprise web application and interactive desktop application domains. There were many performance data visualisation or aggregation techniques from different domains that had fundamentally similar goals. This was notable because we found examples where the research in one domain did not reference or compare their approach to work with similar aspects that was from another domain e.g. the expert system approaches [s56], [s58], [s115], [s172], [s173].

5.3 Evaluating the Utility of Performance Analysis Approaches

Based upon our analysis in Section 4.6 of the evaluation approaches used in the literature we reviewed we have concluded that there has been remarkably little work done to confirm the utility of empirical performance analysis approaches, particularly for large scale object-oriented software. By this we mean very few papers addressed the question of whether the performance feedback being generated for the user was actually useful in helping to improve software performance. In all the papers we included in our review, only four included an evaluation involving human subjects to assess whether the approach was useful for ultimately improving performance. A further 70 gave concrete examples of how using a particular approach led to real performance improvements but few of these examples could convincingly be extrapolated to provide evidence that the approach is useful for object-oriented software in general. Only 42 used real-world examples and only seven of these were from the object-oriented domain (as classified by our Domain facet).

Note that this is not a commentary on the quality of the evaluations done in the individual research papers in our survey. Many papers were presenting a new approach to obtaining a standard set of data with better accuracy or lower overhead, a situation where the utility of the data is assumed to already be proven, and therefore the focus in the evaluation for those papers was in validating the improved accuracy or reduced overheads. Nevertheless there were approaches describing novel types of performance feedback that relied solely on the description of the approach to establish that it was useful. We feel there is a clear gap in the research when it comes to evaluating the utility of performance analysis approaches, particularly for large scale object-oriented software.

5.4 Improving Feedback on the Performance of Large-Scale Object-Oriented Software

The overriding conclusion we have drawn from our study is that there are still opportunities to improve feedback on the performance of large-scale object-oriented software. Whilst there has been research done in this area, which we describe below, there are still possible extensions and innovations to contribute.

We had 253 papers in our study. This included:

- Thiry eight profiling infrastructure papers
- Seventy four papers focussed on some characteristic of performance other than execution cost e.g. memory accesses, request latency or value profiling
- Eighty two that returned only simple feedback i.e. summarised metrics

Therefore only 59 papers describe profiling or analysis approaches investigating execution cost that provide a more advanced (derived or actionable) form of feedback. We describe these papers in more detail below to highlight where we think there are further research opportunities.

Thirty of these papers, whilst applicable to object-oriented software, do not specifically address the challenges provided by large-scale object-oriented software. This means they are likely to struggle to produce useful feedback when used with large-scale object-oriented software. These included:

- Five expert system approaches. These use a rules-based encoding of expert knowledge to classify whether observed behaviour matches known, predefined, undesirable patterns. The major challenge with these approaches is creating suitable expert-system rules. The examples we have seen have been focussed either on HPC applications, with rules mostly focussed around parallel scalability concerns, or component-based (essentially EJB) applications, with rules mostly focussed around poor distributed computing patterns. We did not find any expert system approaches focussed on analysing large-scale object-oriented applications.

The other approaches provide feedback at the method level or basic block level of granularity and so can fail to provide useful feedback when faced with the overwhelming number of fine-grained methods that exist in large-scale object-oriented applications. These consisted of:

- Fourteeen papers that perform a type of automated search for code regions that exhibit certain predefined performance characteristics. These approaches help to rapidly identify specific problem areas, often with a relatively low overhead because of the iterative and targeted search process. Only pre-defined characteristics, such as a poor rate of memory cache hits, can be identified.
- Eight papers covering a range of different approaches to deriving new metrics from the raw measurements that are captured during profiling. Some of these new metrics are built-in advanced metrics that expose a particular characteristic of performance and some allow the user to define their

own metrics using calculations combining existing measurements.
- Three static analysis approaches. Two of these are a type of feedback generated from static optimisers (normally but not necessarily associated with compilers) that indicate where code changes could be made that would enable further static optimisations. The other approach looks for code patterns that equate to known performance bugs. All of these approaches have the typical benefits and weaknesses of static approaches. They can be applied at build time without needing a running environment but are unable to distinguish which code is performance critical.

The remaining 29 papers are those that most directly addressed performance concerns specific to large-scale object-oriented software, and runtime bloat in particular:

- Four papers analyse memory efficiency in Java programs by analysing the allocated objects and structures on the heap. These help give insight into memory bloat but do not directly address issues of execution time performance.
- Seven papers discuss performance data presentation approaches. These techniques assist with interpreting performance data but only help visualise or present the existing data. New insights into the performance behaviour and how the performance could be improved still need to be drawn by the user.
- One static analysis approach, which combines concern input with program analysis for bloat detection. This helps analyse when optional concerns (features) have an impact on the performance of mandatory features. It requires an external source of concern input and does not help identify when the implementation of a mandatory feature is inefficient.
- One paper presents algorithmic profiling, an approach that automatically partitions a program into multiple algorithms and determines a cost function for each of these algorithms in terms of the input size for the algorithm. The focus is on understanding how these algorithms will behave with increasing input sizes. The current implementation is only able to work on algorithms that operate on data structures and has a very high runtime overhead (several orders of magnitude).

The last 16 papers are those that most directly tackle runtime execution bloat:

- Seven approaches focus on identifying object pooling opportunities, these identify object allocation sites that produce many objects that are either very short-lived or very similar or both and as such create repeated and potentially unnecessary work.
- Five papers that focus on tracking possible symptoms of runtime bloat in a data-flow centric manner. They do this by monitoring data copies, reference propagations or the usage of temporary objects and relating them to the code locations that induce them.
- Four approaches focus on the way data structures are created and used, either identifying situations where an implementation is inappropriate for the way it is used or where a data-structure has a low value (is used very little) relative to the cost of its creation.

All of these last 16 papers are focussed on understanding bloat in terms of the inefficient use of data-structures, inefficient data-flows or inefficient creation of objects. In other words they all took a data-flow centric view of the problem, rather than a control-flow centric one. Whilst the majority of the more traditional profilers did take a control-flow centric view of performance, they provided only simple method or calling context summarised metrics. Consequently there is opportunity for providing more advanced control-flow centric feedback on performance of object-oriented software. For example, feedback approaches based on aggregating performance information over the repeated calling patterns that are induced by the design of the software. Additionally the object pooling and data structure based approaches address specific identified performance problems, i.e. object churn and inappropriate data structure respectively, rather than providing general performance feedback. This means that whilst these approaches are useful they are not comprehensive and there are further opportunities for complementary approaches.

6 Threats to Validity

The primary threats to the validity of our results and conclusions are the two classic problems faced by many literature reviews, namely the possibility of researcher bias and the possibility of missing relevant articles. The primary defense against these threats has been to follow a systematic review methodology as we have described in Section 3. This process is designed to be rigorous and objective and relies primarily upon the discipline of defining the protocol to be followed throughout the review before undertaking the review. However it does still rely on subjective interpretation and in our case the risk of researcher bias is increased by the fact that the majority of the work was undertaken by a single researcher, the primary author of this paper. Therefore we took additional steps, which we describe below, at several stages to mitigate this risk.

One of our biggest challenges in conducting this systematic review was the search phase. We experienced a number of challenges similar to those discussed by Brereton et al. [17], in particular the differences in functionality and sophistication between the different mainstream software engineering search engines. As documented in our search results in Section 3.8 each search engine required a different search expression syntax. They also each had different approaches to both stemming (the process of identifying the root of a search term to find relevant partial matches e.g. plurals) and identification of key words for articles, meaning that it was possible to get different results from different search engines even when the same paper was being indexed. However these challenges were minor compared to our biggest challenge, which was generating suitable key phrases for our initial searches. As we discussed when we described our search phrase development in Section 3.4 the

key terms for our domain suffered from having both low precision (terms like 'software' and 'performance' return tens of thousands of results) and low recall (there are many synonyms for terms like 'performance analysis'). We lacked that one key unique domain specific term that would have helped both the precision and recall of our searches.

To mitigate the challenges of the search phase we resorted to being as thorough as possible. We included multiple electronic database searches, manual searches targeted to the venues highlighted as key venues by the results of our electronic searches, and finally referencing checking of included papers. In total we performed automated searches against 6 major electronic databases, manual searches across 66 conference proceedings, and reference checks against many hundreds of papers. In total we manually screened many thousands of papers for inclusion in our survey. Despite this thoroughness we feel it is likely that some relevant articles have been missed. This was highlighted by our manual searches, where a detailed search of five key venues that were already contributing over 60 articles found by our electronic searches, found another 32 articles suitable for inclusion. This worryingly represents an approximately 50 percent miss rate from the electronic searches. More reassuring were the results of the reference snowballing, which resulted in only an additional 19 papers being included, meaning that comparatively few papers of referenceable quality were missed by the earlier searches. Due to the completeness of our searches we believe the number of possibly missed articles is small enough to not substantially affect our conclusions.

Our decision to limit our mapping study only to literature published since January 2000 does mean that we have potentially excluded relevant literature from before this time. However given our particular focus on object-oriented applications, which were still relatively new in 2000, the rapid pace of change in computing since then, and the large numbers of papers we have included in our study, we feel it is unlikely that the excluded papers from before 2000 would significantly impact our conclusions.

To reduce the risk of researcher bias during article selection and data extraction we undertook a process where a random selection of 100 papers returned from the automatic searches was screened for inclusion and then categorised according to the developed review protocol by each of the four authors of this paper. Once each author had completed this process independently the results were compared and any discrepancies were discussed and resolved. The intent of this process was to clarify any ambiguous aspects of the inclusion/exclusion criteria and categorisation process to make them less subjective and easier to apply more consistently. This then allowed us to fine-tune our review protocol before it was finalised and applied to the full review.

The results interpretation process is by its nature a subjective process and is vulnerable to bias or limitations at the earlier stages of the process as missing papers or ambiguous classifications during the data extraction will potentially change the results that are being discussed. One of our primary defenses against these threats was to limit ourselves to analysing and drawing conclusions on clear trends in the data, trends that are not invalidated by small numbers of papers being missed or classified inappropriately. Our results interpretation was also an area where the conclusions were reviewed and discussed amongst the authors of the paper to mitigate the risk of bias from a single researcher.

7 Conclusion

We have presented the details of a systematic mapping study on empirical performance analysis approaches applicable to object-oriented software. We performed a wide-ranging systematic literature search including automated searches of six major software engineering electronic databases and detailed manual searches of five key venues since 2000, screening many thousands of papers, to eventually select 253 articles related to empirical approaches for performance analysis applicable to object-oriented software. We developed a classification scheme (see Fig. 1) to structure and describe literature in the field of empirical software performance analysis and subsequently applied it to our 253 selected articles. This allowed us to give an overview of the current research in the field (see Fig. 2) and examine the trends and gaps in that research. The resulting mapping is useful both for researchers looking to identify new opportunities and related work in this field, and practitioners from industry interested in discovering state-of-art approaches to performance analysis they might wish to apply.

After analysis of our categorisation we came to four key conclusions. Firstly there are relatively few approaches that incorporate the use of any in-depth static information or analysis when analysing performance. There seems to be great scope for leveraging more static analysis to augment the traditional dynamic analysis based approaches. Secondly we found that there are performance analysis approaches from a wide range of different domains that are applicable to object-oriented software. The reverse is likely to be true as well, with object-oriented approaches being increasingly adopted in diverse areas such as HPC and mobile computing the performance analysis approaches developed for object-oriented software are likely to be applicable in those areas in future. Thirdly, there has been little work done evaluating the utility of performance analysis approaches, particularly for large-scale object-oriented software.

Finally we found there are still opportunities to improve on the feedback generated on the performance of large-scale object-oriented software. The majority of existing approaches provide only simple metrics, which are challenging to interpret in large-scale object-oriented software. There is some more recent research, particularly that into runtime bloat, that addresses this problem but to date it has been limited to data-flow, rather than control-flow, based approaches. Despite the importance and prevalence of object-oriented software there are significant challenges in understanding and improving its performance.

Acknowledgments

Mr. David Maplesden is the corresponding author.

References

[1] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," School Comput. Sci. Math., Keele Univ., Staffordshire, U.K., EBSE Tech. Rep. EBSE-2007-01, 2007.

[2] J. Larus, "Spending Moore's dividend," *Commun. ACM*, vol. 52, no. 5, pp. 62–69, 2009.
[3] N. Mitchell, G. Sevitsky, and H. Srinivasan, "Modeling runtime behavior in framework-based applications," in *Proc. 20th Eur. Conf. Object-Oriented Program.*, 2006, vol. 4067, pp. 429–451.
[4] G. Xu, N. Mitchell, M. Arnold, A. Rountev, and G. Sevitsky, "Software bloat analysis: Finding, removing, and preventing performance problems in modern large-scale object-oriented applications," in *Proc. FSE/SDP Workshop Future Softw. Eng. Res.*, 2010, pp. 421–425.
[5] N. Mitchell, E. Schonberg, and G. Sevitsky, "Four trends leading to Java runtime bloat," *IEEE Softw.*, vol. 27, no. 1, pp. 56–63, Jan./Feb. 2010.
[6] M. Woodside, G. Franks, and D. Petriu, "The future of software performance engineering," in *Proc. Future Softw. Eng.*, 2007, pp. 171–187.
[7] C. U. Smith, *Performance Engineering of Software Systems*. Reading, MA, USA: Addison Wesley, 1990.
[8] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 295–310, May 2004.
[9] D. C. D'Elia, C. Demetrescu, and I. Finocchi, "Mining hot calling contexts in small space," in *Proc. 32nd ACM SIGPLAN Conf. Program. Lang. Des. Implementation*, 2011, pp. 516–527.
[10] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Perform. Eval.*, vol. 67, no. 8, pp. 634–658, Aug. 2010.
[11] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proc. 12th Int. Conf. Eval. Assessment Softw. Eng.*, 2008, pp. 68–77.
[12] B. Kitchenham, R. Pretorius, D. Budgen, P. Brereton, M. Turner, M. Niazi, and S. Linkman, "Systematic literature reviews in software engineering—A tertiary study," *Inf. Softw. Technol.*, vol. 52, no. 8, pp. 792–805, Aug. 2010.
[13] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 625–637, Jun. 2011.
[14] T. Dyba, T. Dingsoyr, and G. K. Hanssen, "Applying systematic reviews to diverse study types: An experience report," in *Proc. 1st Int. Symp. Empirical Softw. Eng. Meas.*, Sep. 2007, pp. 225–234.
[15] K. Petersen, "Measuring and predicting software productivity: A systematic map and review," *Inf. Softw. Technol.*, vol. 53, no. 4, pp. 317–343, Apr. 2011.
[16] B. Dufour, B. G. Ryder, and G. Sevitsky, "Blended analysis for performance understanding of framework-based applications," in *Proc. Int. Symp. Softw. Testing Anal.*, 2007, pp. 118–128.
[17] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, Apr. 2007.

David Maplesden received the MSc (Hons) degree in computer science from the University of Auckland, New Zealand, in 2000. He is currently working toward the PhD degree at the Department of Computer Science, University of Auckland. He spent 12 years as a professional software developer both in NZ and overseas mostly building enterprise Java web applications. His current research interests include performance of large scale object-oriented software and tool support for software engineering.

Ewan Tempero received the graduate degree from the University of Otago, New Zealand, with a BSc degree, (honours) in mathematics, in 1983, and the PhD degree in computer science from the University of Washington, in 1990. He is currently an associate professor at the Department of Computer Science, University of Auckland, New Zealand. He has published more than 140 papers in journals and internationally-refereed conferences, mainly in the areas of software reuse, software tools, and software metrics. His current research is developing metrics for measuring the quality of software designs. He is the developer and maintainer of the Qualitas Corpus. He is a member of the IEEE.

John Hosking received the BSc and PhD degrees in physics from the University of Auckland and then joined the Department of Computer Science, University of Auckland, working his way through to full professor before taking up the position of dean at the College of Engineering and Computer Science, Australian National University. He is the dean of Science, University of Auckland. He has recently returned to his new role in Auckland. His research interests include software engineering, visual languages, and knowledge visualisation. He is a member of the IEEE and a fellow of the Royal Society of New Zealand.

John C. Grundy received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. Previously, he was a lecturer and a senior lecturer at the University of Waikato, New Zealand, and a professor of software engineering and the head of electrical and computer engineering at the University of Auckland, New Zealand. He is currently a professor of software engineering and the head of computer science and software engineering at the Swinburne University of Technology, Melbourne, Australia. He is an associate editor of the *IEEE Transactions on Software Engineering*, the *Automated Software Engineering Journal*, and *IEEE Software*. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. He is a member of the IEEE and the IEEE Computer Society.