A Survey of Controlled Experiments in Software Engineering

Dag I.K. Sjøberg, *Member*, *IEEE Computer Society*, Jo E. Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanović, Nils-Kristian Liborg, and Anette C. Rekdal

Abstract—The classical method for identifying cause-effect relationships is to conduct controlled experiments. This paper reports upon the present state of how controlled experiments in software engineering are conducted and the extent to which relevant information is reported. Among the 5,453 scientific articles published in 12 leading software engineering journals and conferences in the decade from 1993 to 2002, 103 articles (1.9 percent) reported controlled experiments in which individuals or teams performed one or more software engineering tasks. This survey quantitatively characterizes the topics of the experiments and their subjects (number of subjects, students versus professionals, recruitment, and rewards for participation), tasks (type of task, duration, and type and size of application) and environments (location, development tools). Furthermore, the survey reports on how internal and external validity is addressed and the extent to which experiments are replicated. The gathered data reflects the relevance of software engineering experiments to industrial practice and the scientific maturity of software engineering research.

Index Terms—Controlled experiments, survey, research methodology, empirical software engineering.

1 Introduction

There is an increasing understanding in the software engineering community that empirical studies are needed to develop or improve processes, methods, and tools for software development and maintenance [6], [4], [35], [16], [43], [5], [32], [41], [50], [15]. An important category of empirical study is that of the controlled experiment, the conducting of which is the classical scientific method for identifying cause-effect relationships.

This paper reports on a survey that quantitatively characterizes the controlled experiments in software engineering published in nine journals and three conference proceedings in the decade from 1993 to 2002. The journals are *ACM Transactions on Software Engineering Methodology* (TOSEM), *Empirical Software Engineering* (EMSE), *IEEE Computer*, *IEEE Software*, *IEEE Transactions on Software Engineering* (TSE), *Information and Software Technology* (IST), *Journal of Systems and Software* (JSS), *Software Maintenance and Evolution* (SME), and *Software: Practice and Experience* (SP&E). The conferences are the International Conference on Software Engineering (ICSE), the IEEE International Symposium on Empirical Software Engineering (ISESE), and the IEEE International Symposium on Software Metrics (METRICS). The conference Empirical Assessment & Evaluation in Software Engineering (EASE) is partially included in that 10 selected articles from EASE appear in special issues of *JSS*, *EMSE*, and *IST*. We consider the above journals to be leaders in software engineering. ICSE is the principal conference in software engineering, and ISESE, Metrics, and EASE are major venues in empirical software engineering that report a relatively high proportion of controlled software engineering experiments.

Research in empirical software engineering should aim to acquire general knowledge about which *technology* (process, method, technique, language, or tool) is useful for *whom* to conduct which (software engineering) *tasks* in which *environments*. Hence, this survey focuses on the kind of technology being studied in the experiments investigated (which reflects the topics of the experiments), the subjects that took part, the tasks they performed, the type of application systems on which these tasks were performed, and the environments in which the experiments were conducted. This survey also includes data on experiment replication and the extent to which internal and external validity is discussed.

The paper is organized as follows: Section 2 describes related work. Section 3 defines the research method for the survey. Section 4 reports the extent of controlled experiments, and Sections 5-10 report our main findings. Section 11 discusses threats to the validity of this survey. Section 12 summarizes.

2 Related Work

Table 1 summarizes the purpose, scope, and extent of four major surveys in addition to this survey. Tichy et al. [43] compare the amount of experimental work published in a few computer science journals and conference proceedings with the amount of experimental work

- *D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, and A. Karahasanović are with the Simula Research Laboratory, Department of Software Engineering, P.O. Box 134, NO-1325 Lysaker, Norway. E-mail: {dagsj, johannay, oveha, vigdis, amela}@simula.no.*
- *N.-K. Liborg is with BNP Paribas, Karl Johansgt. 7, NO-0154 Oslo, Norway. E-mail: nils@liborg.no.*
- *A.C. Rekdal is with Unified Consulting, Ruseløkkveien 14, NO-0251 Oslo, Norway. E-mail: acr@unified.no.*

Manuscript received 6 Jan. 2005; revised 22 June 2005; accepted 27 June 2005; published online 15 Sept. 2005.
Recommended for acceptance by B. Littlewood.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0005-0105.

TABLE 1
Surveys of Empirical Studies in Software Engineering

	(Tichy *et al.* 1995)	(Zelkowitz *et al.* 1997)	(Glass *et al.* 2002)	(Zendler 2001)	Our survey
Purpose	Compares the extent of empirical studies in computer science with other fields	Classifies empirical studies in SE and validates the taxonomy of empirical studies proposed by the authors	Surveys topics, research approaches, research methods, reference disciplines and level of analysis.	Develops a preliminary SE theory from the results of various SE experiments	Surveys topics, subjects, tasks, environments, and internal and external validity of controlled experiments in SE
Scope	Comp. Sci., incl. SE	SE	SE	SE	SE
Journals	ACM (random publications), TSE, PLDI Proc., TOCS, TOPLAS	ICSE Proc., IEEE Software, TSE	IEEE Software, IST, JSS, SP&E, TOSEM, TSE	Various journals and conference proceedings	EASE, EMSE, ICSE, IEEE Computer, IEEE Software, ISESE, IST, JSME, JSS, METRICS, SP&E, TOSEM, TSE
Sampling of papers	1991-1994, one to four volumes per journal, random selection of work published by ACM in 1993	All papers in 1985, 1990 and 1995	Every fifth paper in the period 1995-1999	Not reported	All papers in the period 1993-2002
Number of investigated papers	403	612	369	49 papers assessed, 31 papers analysed in depth	5453 papers scanned, 103 papers analysed in depth

published in a journal on artificial neural network and a journal on optical engineering. In total, 403 articles were surveyed and classified into five categories: *formal theory*, *design and modeling*, *empirical work*, *hypothesis testing*, and *other*. Zelkowitz and Wallace [49] propose a taxonomy of empirical studies in software engineering and report a survey in which 612 articles were classified within this taxonomy. Glass et al. [20] investigate 369 articles with respect to topic, research approach, research method, reference discipline, and level of analysis.

The above surveys give a comprehensive picture of research methods used in software engineering. They differ in purpose, selection criteria, and taxonomies. Nevertheless, their results suggest the same conclusions: The majority of published articles in computer science and software engineering provide little or no empirical validation and the proportion of controlled experiments is particularly low. The surveys propose means to increase the amount of empirical studies and their quality.

The major difference between those surveys and ours is that they describe the extent and characteristics of various types of empirical study, while we provide an in-depth study of controlled experiments only. A comparison of those surveys and ours regarding the extent of controlled experiments is provided in Section 4.

In addition to the general surveys described above, there are several surveys within subdisciplines of software engineering, for example, object-oriented technology [14], testing techniques [28], and software effort estimation [25]. Furthermore, Shaw [38] categorizes the research reported in articles submitted and accepted for ICSE 2002 and Zendler [51] reports a survey of 31 experiments with the aim of developing a preliminary theory about software engineering.

3 Research Method

This section describes the kind of experiments that are considered in this survey and the procedure for identifying and analyzing the relevant articles.

3.1 Controlled Experiments in Software Engineering

Shadish et al. [37] provide the following definitions:

- *Experiment*: A study in which an intervention is deliberately introduced to observe its effects.
- *Randomized experiment*: An experiment in which units are assigned to receive the treatment or an alternative condition by a random process such as the toss of a coin or a table of random numbers.
- *Quasi-Experiment*: An experiment in which units are not assigned to conditions randomly.
- *Correlation study*: Usually synonymous with nonexperimental or observational study; a study that simply observes the size and direction of a relationship among variables.

To identify the effects of the deliberate intervention in an experiment, factors that may influence the outcome, in addition to the treatment, should also be controlled.[1] This is the challenge of internal validity (see Section 10.1). Note that control is not an all or nothing condition; the degree of control varies on a continuous scale.

Based on the definitions given above, we present an operational definition used for this survey. Since the term

1. Some definitions are very explicit on the aspect of control, for example, Zimney [52] defines a psychological experiment as an "objective observation of phenomena which are made to occur in a strictly controlled situation in which one or more factors are varied and the others are kept constant," see discussion of this definition in [10].

TABLE 2
Aspects and Their Attributes for Data Extraction

Aspect	Attributes
Extent	Authors, Affiliation, Country, Year, Journal/Conference.
Topic	Treatment, Title, Keywords.
Subjects	Number of subjects categorised into (subcategories of) students and professionals, Subject selection mode, Subject background and Subject recruitment information (voluntary, part of course, paid, *etc.*).
Task and Environment	Location of experiment, Development tool, Work mode (individual or team), Duration, Application type (commercial or constructed), Application/Task size.
Replication	Replication indicator, *Subjects*, *Topic*, *Extent*.
Internal validity	Category of threat to internal validity, Explicitness.
External validity	Category of threat to external validity, Explicitness.

"experiment" is inconsistently used in the software engineering community (often used synonymously with empirical study), we use the term "controlled experiment:"

Controlled experiment in software engineering (operational definition): A randomized experiment or a quasi-experiment in which individuals or teams (the experimental units) conduct one or more software engineering tasks for the sake of comparing different populations, processes, methods, techniques, languages, or tools (the treatments).

We do not distinguish between randomized experiments and quasi-experiments in this survey because both experimental designs are relevant to empirical software engineering experimentation. Random assignment of experimental units to treatments may not always be feasible, e.g., for logistic reasons. For example, one of the surveyed experiments used units formed from existing training groups in a company—random assignment would, in this case, have disturbed the training process.

We exclude several types of study that share certain characteristics with controlled experiments because, while these may be highly relevant for the field, they do not apply the deliberate intervention essential to controlled experiments. Thus, we exclude correlation studies, studies that are solely based on calculations on existing data (e.g., from data mining), and evaluations of simulated teams based on data for individuals. The last category falls outside our operational definition because the units are constructed after the run of the experiment.

Studies that use projects or companies as treatment groups, in which data is collected at several levels (treatment defined, but no experimental unit defined) are also excluded because we consider these to be multiple case studies [47] (even though the authors might refer to them as experiments). Our survey focuses on articles that provide the main reporting of experiments. This excludes articles that, at the outset, would not provide sufficient data for our analyses (e.g., summaries of research programs). Moreover, usability experiments are not included since we regard those as part of another discipline (human computer interaction).

3.2 Identification of Articles that Report Controlled Experiments

In order to identify and extract controlled experiments, one researcher systematically read the titles and abstracts of 5,453 scientific articles published in the selected journals and conference proceedings for the period 1993-2002. Excluded from the search were editorials, prefaces, article summaries, interviews, news, reviews, correspondence, discussions, comments, reader's letters, and summaries of tutorials, workshops, panels, and poster sessions.

If it was unclear from the title or abstract whether a controlled experiment was described, the entire article was read by both the same researcher and another person in the project team. In the end, 103 articles were selected. Note that identifying the relevant articles is not straightforward because the terminology in this area is confusing. For example, several authors claim that they describe experiments even though no treatment is applied in the study.

3.3 Analysis of the Articles

The survey data is stored in a relational database (MS SQL Server 2000).[2] Some data is specific to an article, some is specific to an experiment, and some information concerns the combination of article and experiment. Moreover, an article might describe several experiments and an experiment might be described in several articles, typically with a different focus in each article. Consequently, we defined a data model with the entities *article*, *experiment*, and *article-experiment* with a set of attributes relevant to our survey. In addition to the survey database, a catalog of all the articles in searchable pdf-format was generated. (About 3/4 of the articles were provided in searchable pdf-format by the journal publishers; the remaining 1/4 were OCR-scanned.)[3] The articles were analyzed according to the six aspects listed in Table 2. Each aspect encompasses a set of attributes for data extraction.

2. MS SQL Server 2000 is a registered trademark of Microsoft Corp.
3. The survey database and catalog of articles may be provided upon request to the corresponding author and under the conditions of a signed agreement toward the use of the data.

TABLE 3
Articles that Report Controlled Experiments

		Articles reporting controlled experiments	
Journal/Conference	**Total no. of articles investigated**	N	Row %
EMSE	124	22	17.7
ISESE	20	3	15.0
METRICS	177	10	5.6
JSS	886	24	2.7
TSE	687	17	2.5
ICSE	520	12	2.3
IST	745	8	1.1
SME	186	2	1.1
IEEE SW	532	4	0.8
TOSEM	125	1	0.8
IEEE Comp	780	0	0
SP&E	671	0	0
All	**5453**	**103**	**1.9**

EMSE started in 1996, ISESE was held only in 2002. METRICS was held in eight of the ten years covered by this survey.

Six researchers analyzed the articles so that each aspect above was covered by at least two persons. After the initial analysis, the results were compared and possible conflicts resolved by reviewing the articles collectively a third time or handing over the article to a third person. The main analysis tool was SAS.[4]

4 Extent

Controlled experiments, as defined in Section 3.1, are reported in 103 (1.9 percent) of the 5,453 articles scanned for this survey, see Table 3. The 103 articles report a total of 113 controlled experiments. Twelve articles report more than one experiment and four experiments are reported in several articles.

EMSE, ISESE, and METRICS, which focus specifically on empirical software engineering, report a higher proportion of controlled experiments than the other journals and the ICSE conference. The mean percentage of controlled experiments across years varies between 0.6 and 3.5, but we see no marked trend over years. An overview of the trend for the individual journals/conferences is presented in the Appendix.

The surveys summarized in Table 1 also report extent. Tichy et al. have the study type definition with the broadest scope and report that 14 percent of the articles published in the specific software engineering journals TSE and TOPLAS (*Transactions on Programming Languages and Systems*) describe empirical work. In Glass et al., the authors classify 3 percent of the articles as laboratory experiments using human subjects and < 1 percent as field experiments. According to the survey by Zelkowitz and Wallace, experiments defined as controlled methods are reported in 2.9 percent of the articles. Our survey finds the lowest percentage of articles (1.9 percent) that report controlled experiments. This might be because our study type definition is narrower than those of the other studies or because our investigation spans more sources and years.

We rank institutions and scholars according to the number of *experiments* published (not the quality), but relative to their fractional representation on the article(s) that reports the experiment. Glass and Chen [18] also ranked institutions and scholars but according to *publication* in systems and software engineering, and they used a more complex ranking scheme for scholars.

In total, 207 scholars are involved in the experiments of our survey. Table 4 presents the top 20 ranked scholars. Due to the fractional distribution, the number of experiments in which a scholar has actually been involved is typically higher than the scores in Table 4. For instance, the top ranked scholar, Giuseppe Visaggio, was involved in six experiments described in four papers authored by one to three scholars, resulting in a fractional score of 4.2 experiments. Among the 20 top ranked scholars, three (Laitenberger, Roper, Wood) were involved in eight experiments, one was involved in seven, four in six, two in five, nine in four, and one was involved in three experiments.

There are 109 institutions from 19 countries involved in the experiments of our survey. The scores for institutions are accumulated from the scores of affiliated authors. Table 5 presents the top 10 ranked institutions.

The institution that has used most professionals as subjects throughout the surveyed time period is Fraunhofer Institute, Kaiserslautern. In total, they used 100 professionals in six experiments, ranging from 11 to 20 in a single experiment. The institution that conducted the experiment involving the largest number (68) of professionals as subjects was Andersen Consulting (now Accenture), Norway.

5 Topics

This section describes two classifications of the 103 analyzed articles according to their main topic. The first classification illustrates the experiments' discipline coverage relative to

4. SAS is a registered trademark of SAS Institute Inc.

TABLE 4
Top 20 Scholars Conducting Controlled Experiments in Software Engineering 1993-2002

Rank	Experiments	Scholar	Affiliation
1	4.2	Visaggio G	Dipartimento di Informatica, University of Bari
2	2.7	Prechelt L	abaXX Technology AG; Fakultät für Informatik, Universität Karlsruhe
3	2.6	Laitenberger O	Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern
3	2.6	Porter A A	Department of Computer Science, University of Maryland
5	2.4	Wohlin C	Dept. of SE and Comp. Sci., Blekinge Inst. of Technology; Dept. of Com. Systems, Lund University
6	2.3	Roper M	Department of Computer Science, University of Strathclyde
6	2.3	Wood M	Department of Computer and Information Sciences, University of Strathclyde
8	2.0	Votta L G	Software Production Research Department, AT&T Bell Laboratories/Lucent Technologies
8	2.0	Koskinen J	Department of Computer Science and Information Systems, University of Jyväskylä
10	1.8	Miller J	Department of Computer Science, University of Strathclyde
10	1.8	Jørgensen M	Department of Informatics, University of Oslo; Simula Research Laboratory, Oslo
10	1.8	Sjøberg D	Department of Informatics, University of Oslo; Simula Research Laboratory, Oslo
13	1.3	El Emam K	Canadian National Research Council, Institute for Information Technology
13	1.3	Regnell B	Department of Communication Systems, Lund University
13	1.3	Höst M	Department of Communication Systems, Lund University
16	1.2	Daly J W	Agilent Technologies, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern
16	1.2	Tichy W F	Fakultät für Informatik, Universität Karlsruhe
16	1.2	Unger B	sd&m GmbH and Co.; Fakultät für Informatik, Universität Karlsruhe
19	1.1	Basili V R	Department of Computer Science, University of Maryland
19	1.1	Lanubile F	Dipartimento di Informatica, University of Bari
…	…	…	…
Total	**113**	**Total number of scholars 207**	

software engineering as a whole, while the second classification has a more technical focus on software engineering method and methodology. The analysis is with respect to article, rather than experiment; this is adequate since no two experiments on different topics are reported in the same article. Both classifications emphasize the treatment of an experiment, since treatment, being the intervention of interest (Section 3), indicates the de facto topic under investigation.

5.1 Classification Scheme: Glass et al.

There are a number of classification schemes for computing science and software engineering, e.g., SWEBOK [1] and Glass et al. [20]. The classification scheme of Glass et al. is aimed at positioning software engineering research relative to a backdrop of overall computing disciplines, i.e., computer science, software engineering, and information systems, and their classification categories are meant to give uniformity across all three fields [19]. The scheme is, therefore, somewhat general. On the other hand, this scheme has actually been used in classifying work undertaken in software engineering and can therefore be used for illustrating the relative topic coverage of controlled experiments in software engineering.

Fig. 1 shows the distribution to topic categories of controlled experiments in software engineering relative to software engineering research in general. Controlled experiments seem to cover at least the categories that are well represented by general SE research, but remember that the overall number of controlled experiments performed is low (Section 2). Recall that experiments on topics purely within human computer interaction are not included in this survey, as is the case for topics purely within information systems. Our focus on experiments with human subjects also excludes a range of software engineering topics.

TABLE 5
Top 10 Institutions Conducting Controlled Experiments in Software Engineering 1993-2002

Rank	Experiments	Institution	Country
1	8.7	Department of Computer and Information Sciences, University of Strathclyde	Scotland
2	7.6	Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern	Germany
3	6.3	Department of Communication Systems, Lund University	Sweden
4	6.2	Department of Computer Science, University of Maryland	USA
5	5.2	Dipartimento di Informatica, University of Bari	Italy
6	4.1	Fakultät für Informatik, Universität Karlsruhe	Germany
7	4.0	Department of Informatics, University of Oslo	Norway
8	2.3	Department of Computer and Information Science, The Ohio State University	USA
9	2.1	Software Production Research Dept., AT&T Bell Labs/Lucent Technologies	USA
10	2.0	Cleveland State University	USA
…	…	…	…
Total	**113**	**Total number of institutions 109 Total number of countries**	**19**

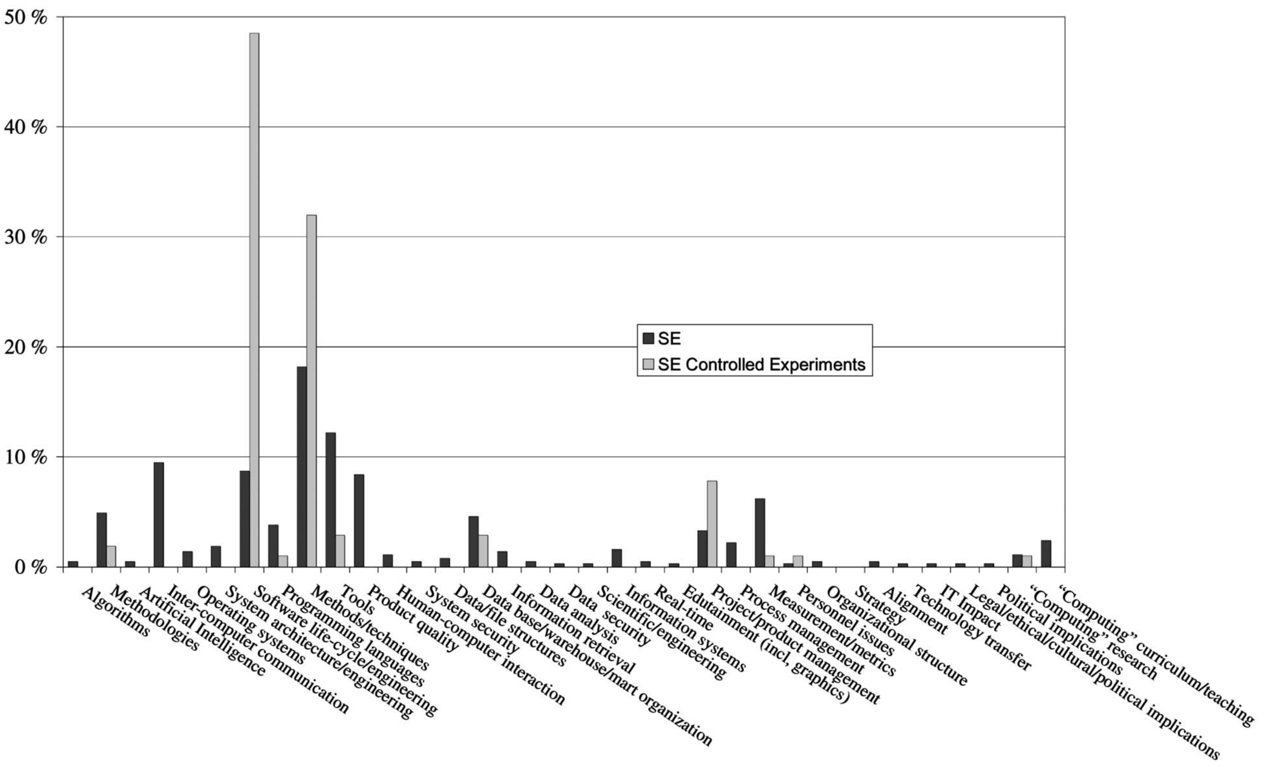


Fig. 1. Comparative distribution to topic of software engineering research and software engineering controlled experiments using the scheme of Glass et al. Only nonvoid categories are shown.

The two most prominent categories are Software life-cycle/engineering (49 percent) and Methods/Techniques (32 percent) due to, respectively, the relatively large number of experiments on inspection techniques and object-oriented design techniques.

5.2 Classification Scheme: IEEE Keyword Taxonomy

The IEEE Keyword Taxonomy [24] provides a more technical perspective than the scheme of Glass et al. [20]. This taxonomy is recommended for authors of IEEE articles, and is an extended version of the ACM Computing Classification [2].

We use the IEEE keywords to denote topic categories. The classification according to the IEEE Keyword Taxonomy is given in Table 6. The two prominent technical areas are *Code inspections and walkthroughs* (35 percent) and *Object-oriented design methods* (8 percent). The numbers of experiments are limited for other areas.

6 Subjects

This section describes the kind of subjects that take part in the experiments, the kind of information that is reported about them, and how they are recruited.

6.1 Number and Categories of Subjects in the Experiments

In total, 5,488 subjects took part in the 113 experiments investigated in this survey. Eighty-seven percent were students and nine percent were professionals. The reported subject types are divided into the categories given in Table 7.

The number of participants per experiment ranges from four to 266, with a mean value of 48.6 (Table 8). Students participated in 91 (81 percent) of the experiments, either alone or together with professionals and/or scientists, and professionals took part in 27 experiments (24 percent). The use of professionals as subjects has been relatively stable over time. Undergraduates are used much more often than graduate students. For one experiment, no information about subject type was provided; for eight experiments, no details about type of students were given; and for five experiments with mixed types of subject, no information about the number in each category was provided.

The issue of differences between students and professionals has been discussed in the literature [13], [12], [48], [36]. Interestingly, while seven articles describe experiments using both students and professionals, only three of them measure the difference in performance between the two groups. In the first experiment, categorized as *Software psychology*, three programming tasks were performed. For two of the tasks, there was no difference between the groups, whereas, for the third task, the professionals were significantly better. In the second experiment, also in *Software psychology*, there was no difference. In the third experiment, categorized as *Maintenance process*, the professionals were significantly better.

The performance in an experiment may differ between subcategories of subjects, that is, there may be an interaction effect between treatment and subcategory [3]. However, none of the surveyed experiments distinguished between subcategories of professionals or students.

The experiment with the highest number of professionals (68) was classified as *Cost estimation* in the IEEE-taxonomy

TABLE 6
Classification of Articles According to IEEE Taxonomy

	Articles			
IEEE Taxonomy	**N (group)**	**N**	**% (group)**	**%**
General	**3**		**2.9**	
Software psychology		3		2.9
Requirements/Specifications	**4**		**3.9**	
Languages		1		1.0
Methodologies		2		1.9
Validation		1		1.0
Design Tools and Techniques	**1**		**1.0**	
Modules and interfaces		1		1.0
Coding Tools and Techniques	**2**		**1.9**	
Object-oriented programming		1		1.0
Structured programming		1		1.0
Software/Program Verification	**3**		**2.9**	
Formal methods		3		2.9
Testing and Debugging	**40**		**35.4**	
Code inspections and walkthroughs		37		35.9
Debugging aids		1		1.0
Testing strategies		1		1.0
Testing tools		1		1.0
Programming Environments/Construction Tools	**2**		**1.9**	
Graphical environments		2		1.9
Distribution, Maintenance, and Enhancement	**3**		**2.9**	
Documentation		1		1.0
Maintenance process		2		1.9
Metrics/Measurement	**1**		**1.0**	
Complexity measures		1		1.0
Management	**8**		**7.1**	
Cost estimation		1		1.0
Productivity		1		1.0
Programming teams		1		1.0
Project control & modeling		1		1.0
Risk management		1		1.0
Time estimation		3		2.9
Design	**15**		**13.3**	
Design notations and documentation		2		1.9
Representation		2		1.9
Methodologies		3		2.9
Object-oriented design methods		8		7.8
Software Architectures	**7**		**6.2**	
Domain-specific architectures		3		2.9
Languages		2		1.9
Patterns		2		1.9
Reusable Software	**4**		**3.9**	
Reuse models		4		3.9
Software and System Safety	**1**		**1.0**	
Software and System Safety		1		1.0
Software Construction	**4**		**3.9**	
Error processing		1		1.0
Programming paradigms		3		2.9
Software Engineering Process	**5**		**4.9**	
Software process models		5		4.9
Total		**103**		**100**

(Table 6). Then, there were five experiments with 29-35 professionals, of which one also employed 20 students. These five were categorized in descending order (regarding number of subjects) as *Modules and Interfaces*, *Code Inspections and walkthroughs*, *Maintenance process*, *Software Psychology* (understanding code), and *Patterns*.

The total number of participants was reported in all the articles, either explicitly or implicitly; in the latter case, we could roughly calculate the number (for instance, from the information that 10 teams averaging four subjects participated). Subject mortality (drop-outs) was reported in 24 experiments (2 percent mortality on average). Even in experiments with as many as 266 subjects (as well as many other experiments with a relatively high number of subjects), no mortality was reported. One article states that *"Non-random drop-out of subjects has been avoided by the experimental design, i.e. assignment of groups only on the second day of the experiment, i.e., directly before the treatment, and not before the pre-test already on the first day of the experiment."* However, most articles say nothing about how mortality was managed.

There are good reasons for conducting experiments with students as subjects, for example, for testing experimental design and initial hypotheses, or for educational purposes [42]. Depending on the actual experiment, students *may* also be representative of junior/inexperienced professionals. However, the low proportion of professionals used in software engineering experiments reduces experimental realism, which in turn may inhibit the understanding of industrial software processes and, consequently, technology

TABLE 7
Subject Categories

Subject Category	Reported Subject Types	N	%
Undergraduates	Undergraduates, Bachelors, Third and fourth-year students, Last-year students, Honors and Majors.	2969	54.1
Graduates	Graduate students, Students following graduate courses or Master's programs, MSc and PhD students.	594	10.8
Students, type unknown	Students in computer science, Students.	1203	21.9
Professionals	Developers, Practitioners, Software engineers, Analysts, Domain experts, Business managers, Facilitators, Professionals.	517	9.4
Scientists	Professors, Post-doctorates, Staff members of educational institutions.	74	1.3
Unknown		131	2.3
Total		**5488**	**100**

transfer from the research community to industry. Hence, to break the trend of few professionals as subjects, new strategies are needed to overcome these challenges, see e.g., discussions in [39], [40].

6.2 Information about Subjects

In order to generalize from an experiment with a given group of subjects (sample population), one needs information about various characteristics and their variation both in the sample and in the group to which the results will be generalized (target population) [7]. For professionals, depending on what we wish to study, it would be relevant to know the variations regarding competence, productivity, education, experience (including domains), age, culture, etc. However, there is no generally accepted set of background variables for guiding data collection in a given type of study, simply because the software engineering community does not know which variables are the important ones. We have chosen to focus on the variables that are reported in the analyzed articles, that is, gender, age, education, experience, and task-related training.

The analyzed articles vary to a large extent on how they report such information. For 14 of the 113 experiments, no information about the subjects was reported. Moreover, the level of detail reported varies substantially. An example of detailed information on programming experience is: "*On average, subjects' previous programming experience was 7.5 years, using 4.6 different programming languages with a largest program of 3510 LOC. Before the course, 69 percent of the subjects had some previous experience with object-oriented programming, 58 percent with programming GUIs.*" An example of a high-level description without figures is: "*Some of the students had industrial programming experience.*"

For the 91 experiments with students, the following information was reported: gender (seven experiments), age (six experiments), grades (six experiments), programming experience (general description: 17 experiments, number of years/languages: 11 experiments), work experience in industry (general description: nine experiments, number of years: nine experiments), task-related experience (64 experiments), and task-related training (27 experiments). The training was either tailored specifically for the experiment or was part of a course, or the experiment could be conducted as part of a training session.

For the 27 experiments with professionals, more details on the subjects' background were given. Categories of professionals such as reviewers, analysts, programmers, and managers were given for seven experiments. Subjects'

TABLE 8
Participants in Experiments

		Experiments		Subjects					
Category of subjects		N	%	Mean	Std	Min	Median	Max	Sum
Students only	Undergraduates only	43	38.1	63.2	61.1	10	43	266	2719
	Graduates only	15	13.3	25.1	11.1	9	24	48	377
	Undergraduates and graduates	16	14.2	60.6	57.8	6	42	208	970
	Students, type unknown	8	7.1	65.5	70.3	13	43	231	524
		82	72.6	56.0	56.8	6	36	266	4590
Professionals only		21	18.6	20.0	14.0	4	20	68	420
Mixed group of subjects		9	8.0	49.3	37.2	12	42	120	444
Unknown		1	0.9	34.0	-	34	34	34	34
Total		**113**	**100**	**48.6**	**51.6**	**4**	**30**	**266**	**5488**

Number and size of experiments in terms of subjects. The mixed group of subjects includes students with scientists and/or professionals.

TABLE 9
Subject Reward Data

	Experiment		Participant	
Reward	**N**	**%**	**N**	**%**
Grade	10	8.8	732	13.3
Extra credits	9	8.0	660	12.0
Payment	3	2.7	121	2.2
Other rewards	1	0.9	24	0.4
No reward	16	14.4	458	8.3
Unknown	74	65.5	3493	64.6
Total	**113**	**100**	**5488**	**100**

degrees were described for three experiments. Gender and age were given for, respectively, two and three experiments. Language and nationality were given for one experiment (subjects from two countries participated). A general description of programming experience was given for two experiments. Programming experience in years/languages was given for seven experiments. A self-assessment of programming experience was reported for two experiments. Work experience in years was given for five experiments. A general description of task-related experience was reported in one experiment. Task-related experience was measured in years for 13 experiments. Task-related training was reported for 12 experiments.

The relatively low and arbitrary reporting on context variables is a hindrance for metastudies, which are needed to identify which context factors influence which kinds of performance. The impact of the various context factors will, of course, depend on the treatments and actual tasks to be performed in the experiments. Future work should investigate the extent to which the variation in performance of subjects can be explained by their background, such as education and work experience, and to increase our knowledge of the impact of using students versus professionals as subjects in software engineering experiments.

6.3 Recruitment of Subjects

Recruiting subjects to experiments is not a trivial task; either from a methodological or a practical point of view. For example, volunteers may bias the results because they are often more motivated, skilled, etc., than subjects who take part because it is mandatory in some way [8]. Information about whether participation was mandatory is reported for 41 (36 percent) of the experiments. For 12 of them (all student experiments), participation was mandatory. Information about subject compensation for taking part in experiments is reported for 39 (35 percent) of the experiments. The grades of students were affected by the participation in 10 cases and they received extra credits in nine cases (Table 9). In three cases, students were paid to take part and, in one case, students were sponsored for a trip to an exhibition. No compensation to professionals is reported. Typically, the experiments with professionals were organized as part of normal projects or training programs and payment was thus implicitly provided by the employer. Hence, it seems that none of the researchers or research teams paid companies or professionals for taking part in experiments.

If one applies statistical hypothesis testing, a requirement is to have a well-defined population from which the sample is drawn: "*If you cannot define the population from which your subjects/objects are drawn, it is not possible to draw any inference from the results of your experiment*" [30].[5] Nevertheless, only one of the experiments in this survey that apply statistical hypothesis testing actually reported sampling from a well-defined target population.

For only a couple of experiments, random sampling of subjects was claimed. How the random sampling was carried out was not described. The dominant approach was convenience sampling: "*Subjects are selected because of their convenient accessibility to the researcher. These subjects are chosen simply because they are the easiest to obtain for the study. This technique is easy, fast and usually the least expensive and troublesome. ... The criticism of this technique is that bias is introduced into the sample.*" [34]. This does not mean that convenience sampling is generally inappropriate. For example, Ferber [17] refers to the *exploratory*, the *illustrative*, and the *clinical* situations in which convenience sampling may be appropriate. In software engineering, the most convenient way of recruiting subjects is to use the students that are taught by the researcher. (Note that convenience sampling is also common in other disciplines such as clinical medicine [34] and social sciences [33].)

To increase the potential for sampling subjects from a well-defined population and to alleviate the problem of having few professionals as subjects (Section 6.1), the experimental software engineering community should apply new incentives, for example, paying companies directly for the hours spent on an experiment [3] or offer the companies tailored, internal courses where the course exercises can be used in experiments [27]. Payment would require that researchers include expenses for this kind of experiment in their applications to funding bodies, see further discussion in [39].

7 Tasks

The tasks that subjects are asked to carry out are an important characteristic of a software engineering experiment. Tasks may include building a software application from scratch or performing various operations on an existing application. This section reports on the surveyed experiments according to a high-level categorization of their tasks and the duration of those tasks. Moreover, we describe the total *magnitude* of the experiments by reporting the product of the number of subjects and the duration of the tasks. Finally, we describe the kinds and size of application and materials used in the experiments.

7.1 Task Categorization

We categorize tasks given to subjects according to the main tasks in a software process. We have defined four general categories, *Plan, Create, Modify,* and *Analyze* that reflect major tasks on software artefacts. Table 10 shows subcategories within these major categories that have been identified in the surveyed experiments.

5. This claim should, as we understand it, not be interpreted outside the context of statistical hypothesis testing. Obviously, even a study without a well-defined population (but with a well-defined sample) may enable the researcher to infer about similar projects, e.g., based on argumentation by analogy or by theory, see further discussion in [26].

TABLE 10
Task Categorization, Duration, and Materials Size

				Duration						Materials[§]
	Experiments			Subject level			Slot level			
Task category	**N[*]**	**%**	**Occ.[†]**	**N[*]**	**Occ.[†]**	*median (h)*[‡]	**N[*]**	**Occ.[†]**	*median (h)*[‡]	**Occ.[†]**
Plan	**11.0**	**9.7**	**11**							
Project planning	4.5	4.0	5				1.5	2	0.5	3
Requirements analysis	1.0	0.9	1				1.0	1	0.7	
Estimation	5.5	4.9	6				0.5	1	0.5	3
Create	**22.8**	**20.2**	**25**							
Design	7.4	6.6	11	2.8	4	0.91	3.8	5	1.0	6
Coding	15.4	13.6	19	4.8	6	3.53	1.3	2	0.9	1
Modify	**18.6**	**16.5**	**22**							
Maintenance -	(18.6)	(16.5)	(22)							(15)
Change design	1.3	1.2	3	0.5	1	0.50	0.5	1	1.0	3
Change code	17.3	15.3	19	12.3	13	0.92	1.5	2	1.7	12
Analyse	**60.7**	**53.7**	**98**							
Inspection -	(35.1)	(31.1)	(37)							(28)
Individual	21.4	19.0	36	6.3	8	2.29	8.0	14	2.0	27
Team	13.7	12.1	29	1.3	3	1.00	6.0	12	2.0	24
Testing	6.6	5.9	10	4.0	6	0.99	1.0	1	0.3	7
Document compreh. -	(19.0)	(16.8)	(23)							(17)
Process doc.	1.0	0.9	1							
Req. doc.	1.0	0.9	1	1.0	1	1.01				1
Design doc.	6.8	6.0	9	3.5	4	0.37	1.0	2	1.5	7
Code doc.	10.2	9.0	12	4.3	5	0.06	1.8	3	2.1	9
All experiments	**113**	**100**	**-**	**41**	**-**	**1.03[*]**	**28**	**-**	**2.0[*]**	**-**

[*] *The fraction of experiments.*
[†] *Occurrences of experiments. One experiment might be represented in several task categories.*
[‡] *Median duration of tasks by category. The last row shows the median total duration for all the tasks of an experiment.*
[§] *The occurrences of experiments in each task category that report size of materials. The total number of experiments that report size of materials is 67.*

Task categorization is somewhat different from topic categorization. Tasks of a certain category can be used to test hypotheses within various topics. For example, a maintenance task can be used to test a certain design or an experiment assigned to the *Patterns* category in the IEEE taxonomy might have design, coding, or maintenance tasks.

Table 10 shows the number of experiments deploying each kind of task. Note that tasks of several categories might be involved in a single experiment. A task is represented by its fraction of all tasks in an experiment, for example, an experiment with one *Design* task and one *Coding* task gives a contribution of 0.5 to each of the two task categories. (Note also that we do not distinguish between tasks and subtasks because there is no commonly agreed definition of the unit of task. Hence, in the relatively few cases in which the experimenters have divided their tasks into subtasks, we have considered them as one task as long as they fall within the same category.) Due to experiment design, a task may be performed several times by the same subjects but with different treatments. In such cases, however, the task is only counted once.

The proportion of planning, creation, modification, and analysis tasks is, respectively, 10 percent, 20 percent, 16 percent, and 54 percent. *Inspection* tasks occur in 37 (33 percent) of the experiments, and are by far the most prominent. This is in accordance with the topic classification of articles reported in Section 5. Thirty-six of these experiments involve individual inspections, 29 involve team inspections. Twenty-eight experiments involve both individual and team inspections. Inspection tasks are typically conducted using pen and paper, although some use support tools.

Document comprehension tasks form the basis of various software engineering tasks and typically involve answering questions about system structure and functionality. Twenty-three experiments involve document comprehension tasks, 12 of these pertain to code documents, nine are design comprehension, one concerns a requirements document, and one concerns process components.

Maintenance tasks pertain to all document types. The surveyed experiments, however, only deal with design and code maintenance tasks; 19 experiments give code maintenance tasks and three give change tasks on design documents (including impact analyses). None give both. In 10 of the code maintenance tasks, new functionality was added. One of the code maintenance tasks is a pen-and-paper maintenance task performed jointly with a comprehension task.

Coding and *Design* are tasks in which new artefacts are produced. Modifying existing code or design documents is classified as maintenance.

Most of the *Testing* tasks involve the generation of test harnesses and test cases. Testing here also includes debugging using debugging tools, but excludes inspections.

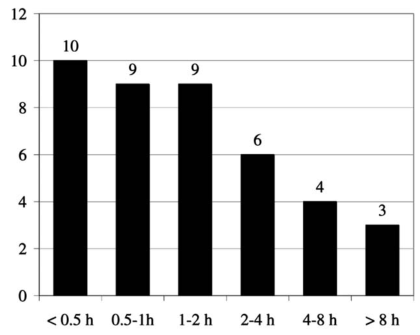


Fig. 2. Distribution of experiments with subject-level duration data to time intervals.

Three experiments investigate the effects of preplanning estimates on detailed estimates (anchoring). In one of these, the *Estimation* task is part of a *Project planning* exercise. One experiment involves estimation in a larger project, although project planning as such is not a task in the experiment in question. Two experiments issue estimation tasks in order to compare various estimation techniques.

Four of the five experiments with *Project planning* tasks are all-student experiments in which the subjects were asked to role-play in project planning or to simulate projects. The fifth one involves both professionals and students assessing how 10 different factors affect the lead-time of software development projects.

In the experiment involving *Requirements analysis*, the subjects were asked to negotiate a software requirements meeting with a customer.

Forty experiments deploy tasks in several categories. Among these experiments, five involve three tasks, two involve four tasks: one has comprehension, maintenance, and inspection (individual and team) tasks, and one has design, coding, team inspection, and testing.

7.2 Task Duration

An important task characteristic is duration. Accurate duration data per subject (typically in dependent variables) is reported in 41 (36 percent) of the experiments and at *slot level* in 28 (25 percent) of the experiments. (Time slots are coarse-grained indications, typically upper bounds, of how much time the subjects took to perform a task. For example, we chose a slot of two hours from the information that "*We gave each subject up to three hours to review each document (i.e., one document in the morning, and one in the afternoon). Only one subject took more than two hours.*") Duration data that is not considered sufficient for analysis is contained in phrases like "*Six days, no time limit,*" "*From 1 to 11 days depending on the subjects' skills,*" and "*Non-programming subjects had 30 min. to finish their task. Programming subjects had one week*".

Fig. 2 shows the frequency by time interval of the 41 experiments with detailed subject-level time information. It appears that about 2/3 of those experiments last less than two hours.

The two leftmost "Subject level" columns of Table 10 show, for each task category respectively, the fraction of and the number of experiments with subject-level duration data that include tasks of this category. The third "Subject level" column shows the median duration in hours for these tasks. For example, three experiments have duration data for design tasks. The fraction of the time spent on design activities in these three experiments, relative to the total time for all experiments (with subject-level time data), is 2.3. The median time used on design tasks in these experiments is 0.85 hours. The median duration of all experiments with subject-level data is 1.0 hours and 2.0 hours for the experiments with slot-level data.

Fig. 3 shows the actual task duration for the subject-level occurrences of Table 10. In the interest of saving space, four data points at, respectively, 25 (*Change code*), 18.5 (*Coding*), 18.5 (*Design*), and 55 hours (*Coding*) are omitted from the

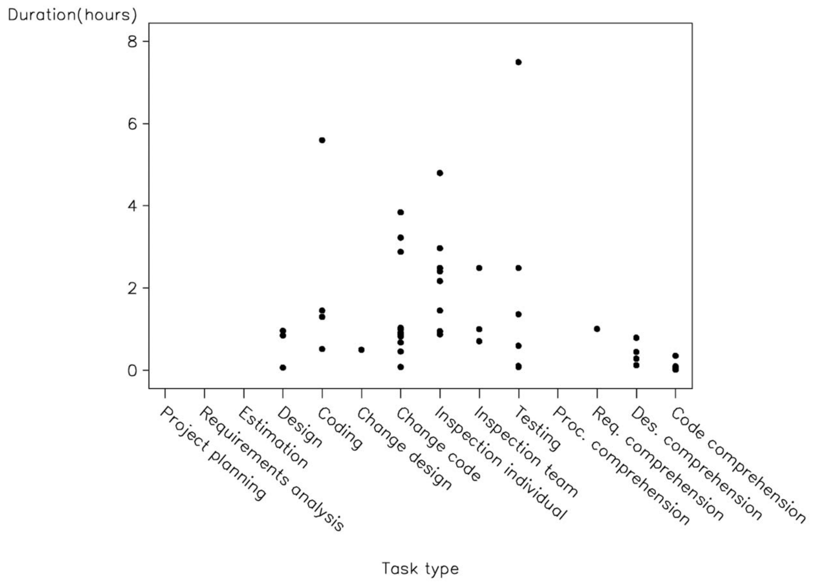


Fig. 3. Task categories and subject-level duration data.

TABLE 11
Distribution of Experiments to Subject Number, Duration, and Subject-Duration Categories

Measure	**Subjects** (N)			**Duration** (*h*)			**Person-Hours**		
Level	S	M	L	S	M	L	S	M	L
Students (only)	15	18	21	18	19	17	14	24	16
Professionals (only)	5	3	0	3	2	3	4	3	1
Combination/Other	1	3	3	2	2	3	2	2	3
Total	**21**	**24**	**24**	**23**	**23**	**23**	**20**	**29**	**20**
Sum			**69**			**69**			**69**

figure. It appears that there is large variance in duration and that it seems independent of the type of task being performed.

Little is mentioned in the articles about control over context variables in experiments with multiple sessions, idle periods, or that span several days or weeks. Although the issue in principle concerns all experiments, it would be particularly interesting to know how experimenters have ensured control in experiments that involve long tasks.

The data described above reflects the duration of explicitly measured software engineering-specific tasks as described in Section 7.1. Often, however, subjects perform additional tasks (training, preparation, postmortem questionnaires, etc.) whose durations are not captured in dependent variables or are otherwise measured explicitly. If one wants to reflect the total time spent by subjects (perhaps in the interest of logistics), information at a different level must be gathered. Although most experiments (close to 80 percent) provide some sort of information about total experiment duration, the data is, in general, measured and reported arbitrarily and is consequently difficult to summarize here.

The median duration of the tasks of 1.0/2.0 hours is, of course, a very small fraction of the time of a typical industrial development project. The extent to which short tasks are a threat to external validity is difficult to judge in general. The actual tasks in the experiments *may* be representative of typical industrial (sub)tasks. However, the lack of studies that describe "typical" tasks within certain categories and contexts makes such a judgment difficult. More studies are needed to investigate the relevance of the tasks being conducted in software engineering experiments.

7.3 Magnitude of Experiments—Combination of Number of Subjects and Duration

Many aspects of the complexity of software engineering only manifest themselves in controlled experiments if the experiments involve a sufficiently large number of subjects and tasks, for example, differences among subgroups of subjects [3]. Hence, we can characterize the experiments in terms of the scale of the combination of subjects and tasks (here, measured in terms of duration of the task). The magnitude of an experiment can be described in terms of the total number of person-hours or person-days; that is, the number of subjects multiplied with the length of the tasks.

In this survey, the experiment with the largest number of professionals lasts less than one hour. However, in general, there seems to be no significant relationship between duration and the number of subjects.

We here categorize the 69 experiments with duration data (41 with subject-level data and 28 with slot-level data), according to subject numbers and task duration into, respectively, S (small), M (medium), and L (large), such that each category contains roughly 1/3 of the experiments. In practice, this gives the following categories: For subject numbers, S: $\leq$ 23, M: 23-47, and L: $>$ 47. For duration, S: $\leq$ 0.96 hours, M: 0.96-2.9, and L: $>$ 2.9 hours. (The subject groups cannot be made completely even because there are six experiments with 24 subjects.) The person-hours categorization is obtained by crossing these two categorizations in configurations (subject category, duration category) as follows: S (small): (S,S), (S,M), (M,S); M (medium): (S,L), (M,M), (L,S); L (large): (M,L), (L,L), (L,M). Table 11 shows that experiments with professionals use a smaller number of subjects than do experiments with students. Both experiments with students and experiments with professionals have a uniform distribution for the three levels of duration. Regarding magnitude, most student experiments are in the middle category and a fair number are large, while most experiments with professionals are small and only one experiment is large.

7.4 Application and Materials

Applications may be of various types, such as commercial, open source, the result of a student project, or custom-built for the purpose of the experiment. Table 12 shows that 75 percent of the surveyed experiments involved applications that were either constructed for the purpose of the experiment or were parts of student projects. Commercial applications were used in 16 experiments, of which 10 included inspection tasks (eight of these had team inspections in addition to individual inspections), two included design tasks, one had coding and maintenance (change code), one had coding only, one had

TABLE 12
Distribution of Experiments to Application Type

Application type	**N**	**%**
Constructed	80	70.8
Commercial	16	14.2
Student project	5	4.4
Open source	0	0.0
Unclear	12	10.6
Total	**113**	**100**

(design) comprehension and maintenance (change design), and one had estimation. For 12 experiments, the reporting is unclear in this respect, but 11 of these appear to have used custom-built applications. There are no open-source applications registered in this survey. The small fraction of commercial or industrial applications used in current software engineering experiments puts in question the possibility of generalizing the experimental results to an industrial setting.

The size of the materials presented to subjects gives some indications of the comprehensiveness of the experiments. Size in the form of pages, lines of code (LOC), or other quantities is reported in 67 (59 percent) of the experiments. The diversity of the surveyed experiments and how they report information about materials makes it difficult to give a systematic overview of the size of the experiment materials. Nevertheless, below, we describe in brief the size of materials per task category, cf. the rightmost column of Table 10.

Three experiments with *Project planning* tasks report materials size: a one-page case scenario, a scenario in terms of 500 adjusted function points, and a four-page program representation, respectively. The three *Estimation* experiments are based on a 1,000 person-hour project, a four-page program representation, and on the creation of 10 programs (no sizes on these), respectively.

Five of the six experiments with size indications for *Design* tasks are on requirements documents (1-2 pages, six modules). The sixth experiment gives a one-page task description. Materials size is reported in one instance for a coding task (specification given in three tables).

Two experiments with *Change design* tasks have a 30 page design document as material (one experiment is an internal replication of the other; the materials of the former are improved, but their sizes are more or less the same), and one has a 1,500 LOC system as input. In the experiments with *Change code* tasks, the applications to be maintained range from 54 to 2,700 LOC. The largest application also involves 100 pages of documentation. Three experiments report the number of classes (6-10).

Twenty-eight experiments give materials size for *Inspection* tasks (individual or team). Fourteen give LOC (ranging from 135-3,955 LOC). In one instance, the materials size is given as 300 LOC, but the size of the entire system is 65,000 LOC. Page counts (ranging from 16-47 pages) are given in 14 instances (all different from the 14 with LOC). Materials size for *Testing* tasks (25-2,000 LOC) is reported in seven experiments (one also reports 10 classes). Reported materials sizes for *Document comprehension* tasks are varied (five diagrams, seven screens, 16 screenshots, etc.), but five experiments give LOC (92-2700 LOC) for *Code comprehension* tasks and five experiments give page counts (2-30 pages) for *Design comprehension* tasks.

In addition, some experiments (with tasks in the *Create* category) report the size of produced task solutions. Five experiments with *Coding* give LOC (in the range 86-2,000 LOC) for produced code, and in one experiment the size for a *Design* task is provided implicitly, in that the solution design document written by the experimenters is two pages. Also, the amount of added code is given for two maintenance tasks: 50-150 LOC and 35-79 LOC, respectively.

8 Environments

The strength of controlled experiments lies in that they may be used to isolate causal relationships. However, controlled experiments in the field of software engineering are often conducted in artificially designed environments that make it difficult to generalize the results to industrial contexts. In short, "Internal and external validity can be negatively related" [37]. This section describes the surveyed experiments according to their location and tools used.

8.1 Location

There is a trade-off between realism and control regarding the location of an experiment. Running an experiment in the usual office environment of subjects that are professionals allows a certain amount of realism, yet increases the threat to internal validity due to breaks, phone calls, and other interruptions. Controlling and monitoring the experiment is easier in a laboratory set up but, in such a setting, realism suffers.

For the 27 experiments with professionals or with professionals and students, 17 report no explicit information about the experimental setting. Only one experiment is reported to have been run in a usual office environment. The pilot of another experiment was run in an office environment, but the main experiment was run in a classroom setting in order to increase the internal validity. Three other experiments were run in a classroom setting, two of which were run as part of a training course. Seven experiments are explicitly reported to have been run in a laboratory environment.

Of the 85 experiments with students or with students and scientists, 56 report no explicit information about the experimental setting. For, respectively, 13 and seven of those experiments, it was explicitly stated that they were conducted in a laboratory and classroom. For another group of nine experiments, some sort of university setting was stated, for example, "*academic settings*," "*conducted under exam conditions*," and "*supervised setting*." However, one may assume that all the experiments with students were carried out in a laboratory or classroom. Moreover, we believe that the distinction between a classroom and laboratory setting for students may be blurred and may depend on cultural differences, apart from the fact that a laboratory usually would include the use of PCs or workstations (2/3 of the experiments that report the use of a laboratory also report the use of PC or workstation, see the next section).

Approximately half of all the experiments with students report the name of the actual university/college. For the 27 experiments that include professionals, the name of the company is reported in 12 cases. For four experiments, the company is not named, but the type of company is specified. Note that, depending on the actual experiment, certain companies have a policy such that they must remain anonymous in the reporting of the experiment. In five cases,

TABLE 13
Distribution of Experiments to Specified Tool

Tool	N	%
PC or workstation (only)	32	28.3
Pen and paper (only)	25	22.1
Combination	5	4.4
Unknown	51	45.1
Total	**113**	**100**

the professionals are described as coming from "several" companies or organizations. The exact number of companies is not given.[6]

8.2 Tools

It is a challenge to configure the experimental environment with an infrastructure with supporting tools that resembles an industrial development environment. Among the surveyed experiments, 55 percent report on tools to support the tasks of the experiments (Table 13). This includes both explicit descriptions, e.g., "*Sun-4, GNU C compiler*," and implicit, but clear indications, e.g., "*Developed programs were run against a set of test data*." Table 13 shows that the use of computer tools is slightly higher than the use of pen and paper. However, it is likely that a larger proportion of those experiments that do not report on tools are actually pen and paper experiments, because the added effort and administrative overhead of using computer tools might inspire researchers to report the use of tools more than the use of pen and paper.

The task types that give the largest and smallest contribution to the *PC or workstation* category are, respectively, *Coding* and *Inspection*. Other than that, there is little correlation between task type and tool for the experiments that actually report on this issue. Moreover, there was no difference between experiments with professionals and experiments with students regarding the use of tools.

Three of the five experiments with *Combination* in Table 13 explicitly test the effects of computerized tool use versus pen and paper.

The relatively meager proportion of experiments that report on the use of tools to support assigned tasks may be due to an unawareness of, or a lack of interest in, the relevance of this issue. For example, most of the experiments in the *Unknown* category are inspection experiments, for which it may be normal to use pen and paper. However, for most design, coding, testing, and maintenance tasks, computer tools would have been used in an industrial setting, although the line is not clear-cut. For example, designers may sketch preliminary versions by hand, but the final design would be made using a tool.

6. Before we decided to rely exclusively on the information reported in the articles, we approached the corresponding authors of these five experiments to acquire more information about the extent of companies involved in the experiments. It turned out that in two experiments, the subjects attended a course aimed at people from industry (the number of companies of the participants was unknown). One author replied that it was a mistake in the article; all participants actually came from the same company. One replied that he did not know, but our impression was that it was only two companies. The last one did not respond to our request.

In general, increasing the realism of software engineering experiments entails an increased use of industrial supporting tools. The community should thus recognize the effort and resources needed to set up PC or workstation environments with the right licenses, installations, access rights, etc., and to familiarize the subjects with the tools. Moreover, the tools must be checked to demonstrate acceptable performance and stability when many subjects are working simultaneously.

In the experiments of this survey, there is almost no discussion of the relationships among the three dimensions *subject*, *task*, and *environment*. For the community to progress, this issue needs to be investigated. For example, a professional development tool will probably become more useful the larger and more complex the tasks and application systems become, assuming that the subjects are sufficiently proficient with the tool.

9 Replication

In this survey, 20 of the experiments are described by the authors themselves as replications. These experiments constitute 14 series of replications. Table 14 summarizes the series including both the original experiments and the replications and reports differences between them. Most replications (35 percent) are conducted in the area of *Inspection* (seven replications in series 1, 2, and 3) and *Maintenance* (five replications in series 4, 5, and 6). Among the 20 replications, five can be considered as *close* replications in the terminology of Lindsay and Ehrenberg [31], i.e., one attempts to retain, as much as is possible, most of the known conditions of the original experiment. The other replications are considered to be *differentiated* replications, i.e., they involve variations in essential aspects of the experimental conditions. One prominent variation involves conducting the experiment with other kinds of subject; three replications use professionals instead of students, three use undergraduates instead of graduates, and one uses students instead of professionals. Other variations include conducting the experiment on different application systems (four) and with different tasks (three).

In all the five close replications, the results of the original experiment were confirmed (three were conducted by the same authors, two by others). Among the 15 differentiated replications, seven were conducted by other authors. Six of these reported results differing from the original experiment and one partly confirmed the results of the original experiment. Among the differentiated replications conducted by the original authors, we found the opposite pattern; seven replications confirmed the results of the original experiment and only one reported partly different results.

"Methodological authorities generally regard replication, or what is also referred to as "repeating a study," to be a crucial aspect of the scientific method" [31]. However, only 18 percent of the surveyed experiments were replications. A discussion of the form and extent of replication that would benefit software engineering is beyond the scope of this paper, but should be an issue of discussion for the research community.

TABLE 14
Replicated Experiments

Series	Topic	Exp.	Stud.	Prof.	Con.	Rej.	Authors	Repl. Type	Other differences
1	Perspective-Based Reading (requirements inspection)	0	X		-	-	-	-	
		1		X	X		same	differentiated	
		2	X			X	others	differentiated	undergrads, (originally graduates)
		3	X			X	others	differentiated	undergrads, more, time extended
		4	X			X	others	differentiated	undergraduate
2	Perspective-Based Reading	0		X	-	-	-		
		1	X			X	others	differentiated	
3	Perspective-Based reading	0		X	-	-	-		
		1		X	X		same	differentiated	diff. applications
		2		X	X		same	close	
4	Maintenance Process	0	X		-	-	-		
		1	X		X		same	differentiated	More tasks than in Exp. 0
		2		X	X		same	differentiated	Same as Exp. 1
5	Maintainability of OO systems (inheritance depth)	0	X		-	-	-		
		1	X		X		same	close	
6	Maintainability of OO systems (inheritance depth)	0	X		-	-	-		
		1	X			X	others	differentiated	diff. appl. and tasks, added hypotheses
		2	X			X	others	differentiated	diff. appl.
7	Quality guidelines (maintainability of OO systems)	0	X		-		-		
		1	X		X		same	differentiated	more subjects, diff. tasks
8	DB referential integrity metrics	0*	X		-	-	-		
		1		X	X	X	same	differentiated	
9	Layering and encapsulation	0	X		-	-	-		
		1	X		X		same	close	
10	Comprehension of OO models	0	X		-	-	-		
		1	X		X		same	differentiated	diff. applications
11	Visual depiction of decision stmt.	0*		X	-	-	-		
		1		X	X		others	close	
12	Defect detection	0*	X		-	-	-		
		1	X		X		others	close	
13	Use Case guidelines	0*	X		-	-	-		
		1	X		X	X	others	differentiated	diff. eval. criteria
14	Design Patterns	0	X				-		
		1	X		X		same	differentiated	diff. prog. lang. and rating scale.

Column Exp. *presents the number in the replication series. The original experiments are denoted by '0'. Columns* Stud. *and* Prof. *indicate whether the subjects were students or professionals. Columns* Con. *and* Rej. *indicate whether the replications confirm or reject the findings of the original experiment. '*' indicates that the original experiment was published in a journal or conference proceedings not included in this survey.*

10 Threats to Internal and External Validity

Two important aspects of the quality of an experiment are their internal and external validity. This section discusses how, and the extent to which, threats to internal and external validity are reported for the surveyed experiments. Descriptions of such threats are made in various ways and under different headings. For 54 experiments (48 percent of all experiments), there is a special section entitled "Threats to (internal/external) validity" or other combinations that include the terms "threats" or "validity." Nine other experiments (8 percent) have special sections on threats to validity but with other names (e.g., "Limitations to the results"). The reporting of threats to validity in yet another eight experiments were found in other sections.

10.1 Internal Validity

Internal validity of an experiment is "*the validity of inferences about whether observed co-variation between A (the presumed treatment) and B (the presumed outcome) reflects a causal relationship from A to B as those variables were manipulated or measured*" [37]. Changes in B may have causes other than, or alternative to, the manipulation of A. Such an alternative cause for the outcome is called a *confound* or *confounding factor*. For further discussions (including formal definitions) of concepts of confounding, see [21].

Threats to internal validity are addressed explicitly for 71 experiments (63 percent). (We did not include threats that are addressed implicitly as part of the experimental design.) We classified the reporting of threats to internal validity according to the scheme of Shadish et al. [37] shown in Table 15. That table also shows examples of threats in the various categories reported for the surveyed experiments. A version of this scheme, with other examples from software engineering, is presented in [46].

Table 16 shows the distribution of the experiments according to the scheme of Table 15. Almost half of all experiments report on selection threats (46 percent) and/or instrumentation threats (40 percent). The distribution of number of threat categories reported is as follows: 22 experiments report one threat, 11 experiments report two threats, 23 report three, 10 report four, four report five, and one experiment reports seven threats.

Table 16 also shows whether the authors consider the threats to be present but not handled by the authors, or reduced or eliminated due to actions taken by the authors or due to other circumstances. Only 18 percent of the threats (reported in 23 percent of the experiments) are not handled, but it may be the case that threats that are not reduced or eliminated are underreported.

TABLE 15
Threats to Internal Validity: Reasons Why Inferences that the Relationship between Two Variables Is Causal May Be Incorrect

	Description given by Shadish *et al.*	Examples from the survey
1.	*Ambiguous Temporal Precedence*: Lack of clarity about which variable occurred first may yield confusion about which variable is the cause and which is the effect.	None
2.	*Selection*: Systematic differences over conditions in respondent characteristics that could also cause the observed effect.	Random assignment and blocking, in combination with randomisation or alone, and within-subject design were often mentioned as reducing factors.
3.	*History*: Events occurring concurrently with treatment could cause the observed effect.	Most cases concerned individuals or teams communicating during the experiments. Attempts to reduce this effect include: "The subjects were instructed not to discuss the experiment or otherwise do anything between the tests that could cause an unwanted effect on the results."
4.	*Maturation*: Naturally occurring changes over time could be confused with a treatment effect.	Most cases concerned boredom, fatigue, demotivation and loss of enthusiasm, for example: "The boredom effect might have affected the second run of the experiment, because subjects had to perform a second complete inspection using the same review technique", "Demotivation may also play a part as subjects become bored with three weeks of testing"*
5.	*Regression*: When units are selected for their extreme scores, they will often have less extreme scores on other variables, an occurrence that can be confused with a treatment effect.	"The absence of pretest scores to assign subjects to groups, the use of simple tasks, and the presence of multiple groups control for statistical regression"
6.	*Attrition*: Loss of respondents to treatment or to measurement can produce artifactual effects if that loss is systematically correlated with conditions.	"A threat to the internal validity that was considered in the analysis is that the subjects did not have enough time to apply all the use cases", "Of the twenty subjects who expressed an interest in the study only thirteen of them actually turned up to participate"
7.	*Testing*: Exposure to a test can affect scores on subsequent exposures to that test, an occurrence that can be confused with a treatment effect.	"We cannot exclude that learning was still in progress during the experiment. We tried to minimize the learning effect by teaching requirements specification and review and having a training session before the experiment itself."
8.	*Instrumentation*: The nature of a measure may change over time or conditions in a way that could be confused with a treatment effect.	"Instrumentation effects may result from differences in the specification documents. Such variation is impossible to avoid, but we controlled for it by having each team inspect both documents."
9.	*Additive and Interactive Effects of Threats to Internal Validity*: The impact of a threat can be added to that of another threat or may depend on the level of another threat.	None

* *Further discussions of the impact of motivation of subjects in software engineering experiments may be found in* [23].

Classifying internal validity is not always straightforward. For example, "learning effects" are often classified as "maturation" in the experiments, while this should be "testing" according to the categories given in Shadish et al. [37]. Maturation threats refer to "*natural changes that would occur even in the absence of treatment, such as growing older, hungrier, wiser, stronger, or more experienced*," while testing threats refer to effects of practice and familiarity within the experiment that could be mistaken for treatment effects [37]. Moreover, threats that by this scheme pertain to *statistical conclusion validity* or *construct validity* were, for a few experiments, reported as internal validity threats. In part, this may be due to nontrivial subtleties in threat classification, illustrated by the fact that the line of development starting with Campbell et al. [9], via Cook et al. [11] to the present classification scheme in [37], shows considerable variation. For example, the notions of statistical conclusion validity and construct validity appeared for the first time in 1979 [11].[7]

10.2 External Validity

External validity concerns inferences about the extent to which a causal relationship holds over variations in persons, settings, treatments, and outcomes [37]. This section summarizes how the authors report threats to external validity regarding these issues.

Threats to external validity are reported for 78 experiments (69 percent). Table 17 shows a categorization of the

7. In addition, there are threats that scholars put in different main categories. For example, what Trochim [44] and Wohlin et al. [46] refer to as "social threats" are categorized as threats to internal validity by them, but as threats to construct validity by Shadish et al. [37]. Four experiments address "social threats" in our survey, but since we follow the scheme of Shadish et al., such threats are not included in our survey.

TABLE 16
Threats to Internal Validity

	No of experiments				
Category	**Threat not handled**	**Threat reduced**	**Threat eliminated**	**Total**	**% of all experiments**
Selection	10	35	7	**52**	**46.0**
Instrumentation	9	30	6	**45**	**39.8**
Maturation	3	14	6	**23**	**20.4**
Testing	2	22	4	**28**	**24.8**
History	3	9	6	**18**	**15.9**
Attrition	5	3	4	**12**	**10.6**
Regression	0	1	1	**2**	**1.8**
Ambiguous Temporal Precedence	0	0	0	**0**	**0.0**
Additive and Interactive Effects of Threats to Internal Validity	0	0	0	**0**	**0.0**
No of threats*	**32 (17.8%)**	**114 (63.3%)**	**34 (18.9%)**	**180 (100%)**	
No of Experiments	**26 (23.0%)**	**55 (48.7%)**	**19 (16.8%)**	**71† (62.8%)**	

* *We do not distinguish between one or more threats within a category for a given experiment; that is, only one threat per category is counted per experiment.*

† *Note that the total number of experiments is not the sum of the previous three columns because one experiment may be represented in more than one category.*

threats based on Shadish et al. [37]. Threats regarding subjects are discussed in a total of 67 experiments (rows one, four, five, six, and seven), regarding task in 60, environment in 23, and treatment in six.

Most threats regarding subjects deal with difficulties of generalizing from students to professionals (45 experiments). Another category of experiments (14) also uses students, but the experimenters argue that this may not be a threat to validity because the students for this kind of task would (probably) have the same ability as professionals (seven), because the students were close to finalizing their education and starting work in the industry (four), or because one other study showed no difference between students and professionals (three). A few experiments (three) that use professionals claim that threats to external validity were not critical because the experiment was conducted with professionals (they did not discuss the representation of their actual sample of professionals). A few experiments (three) considered that running an experiment within a single organization was a threat to the generalization to other organizations. Very few experiments (two) explicitly described the lack of random sampling as a threat to validity.

Most of the task-related threats concern size and complexity of the tasks (16 experiments) and experimental material (34), such as program code, inspection documents, and database systems. For experiments on inspection, one threat discussed was that the inspection process applied was not considered representative for industrial practice (nine). The (short) duration of the experiment was also regarded as a threat (three). One experiment stated that "*all our results were obtained from one project, in one application domain, using one language and environment, within one software organization. Therefore, we cannot claim that our conclusions have general applicability, until our work has been replicated.*" Another experiment stated that the subjects might not have used the technique intended to be studied in the experiment.

Threats regarding environment were either stated as a problem of generalizing from the experimental setting with no specific reasons (five experiments) or stated with concrete reasons for the difficulties: use of laboratory or classroom (nine), individual work (five), and use of pen and paper (six).

A major finding is that the reporting is vague and unsystematic. The community needs guidelines that

TABLE 17
Threats to External Validity

Factors addressed as threats to external validity	**Experiments**	**%**
Subject (only)	14	12.4
Task (only)	10	8.8
Environment (only)	1	0.9
Subject and environment	2	1.8
Subject and task	31	27.4
Subject, environment and task	14	12.4
Treatment and subject, task or environment	6	5.3
Threats to external validity not addressed	35	31.0
Total	**113**	**100**

provide significant support for how to draw conclusions from the experimental results and on how to address threats to internal and external validity and their consequences.

11 Threats to the Validity of this Survey

The main threats to validity for this study are publication selection bias, inaccuracy in data extraction, and misclassification.

11.1 Selection of Journals and Conferences

We consider the 12 surveyed journals and conferences to be leaders in software engineering in general and empirical software engineering in particular. (Our selection of journals is a superset of those selected by others, as shown in Table 1.) Nevertheless, a systematic survey that included, in addition, gray literature (theses, technical reports, working papers, etc.) describing controlled experiments in software engineering would, in principle, provide more data and allow more general conclusions to be drawn [29].

11.2 Selection of Articles

To help ensure an unbiased selection process, we defined research questions in advance, organized the selection of articles as a multistage process, involved several researchers in this process, and documented the reasons for inclusion/exclusion as suggested in [29].

The initial investigation of the titles and abstracts of 5,453 articles resulted in 140 survey articles. Based on recorded comments, 80 of these were reanalyzed by one or two other researchers and discussed in the project group. Seventeen further articles were then excluded because they described studies without a treatment. Moreover, three articles were found to be exploratory, observational, or constituting a prestudy. Eight were found to fall outside the field of software engineering, five were excluded on the grounds that they were summary articles, while four articles described multiple case studies. We used Inspec and various search engines to check the completeness of our inclusion and cross-checked for inclusion with other surveys [51], [22], [25]. Still, the process was difficult and we may not have managed to detect all articles that we would have liked to include.

Another challenge was that there is no keyword standard that we are aware of that distinguishes between methods in empirical software engineering and that could be used to extract controlled experiments in a consistent manner. For example, none of the selected articles matched the IEEE keyword taxonomy; indeed, this taxonomy has no appropriate keywords for the methods of empirical software engineering. (*MIS Quarterly* has ceased to use their keyword classification scheme due to the presence of full-text search engines and the difficulty of keeping keyword classification schemes up to date [45].)

Moreover, article and experiment duplication is a potential threat to frequency counts and the statistics in this survey. Among the 113 experiments covered in the 103 articles, 109 are reported in one article, two are reported in two articles, one is reported in three articles, and one is reported in four. Among the 103 surveyed articles, 91 report a single experiment, seven report two experiments, and five report three experiments. We detected one case of near article duplicates in different journals. The structure of the database is designed to handle duplication, but a threat would be that duplication goes undetected. However, at least three people have read through all relevant articles without detecting further duplicates.

11.3 Data Extraction

The data was extracted from the articles independently by two researchers. The interrater agreement varied from 73 percent to 100 percent. Disagreements were resolved by discussion and, when necessary, by involving other project members. Data extraction from prose is difficult at the outset and the lack of standard terminology and standards for reporting experiments in software engineering may have resulted in some inaccuracy in the data.

11.4 Classification to Topics

The classification of articles to topics was done in two steps. First, the articles were classified automatically on the basis of title, list of keywords, and registered treatment. Then, this classification was double-checked by two researchers. The interrater agreement between the algorithm and the two researchers was 75 percent for the comparative classification using Glass et al.'s scheme and 66 percent for the IEEE-classification. The topic classification was difficult due to the lack of a well-defined method of classifying according to the schemes used.

12 Summary

This paper reported a survey that characterized quantitatively the controlled experiments in software engineering published in nine journals and three conference proceedings in the decade from 1993 to 2002. Included were randomized experiments or quasi-experiments in which individuals or teams (the experimental units) applied a process, method, technique, language, or tool (the treatments) to conduct one or more software engineering tasks. Out of 5,453 articles scanned, we identified 103 articles that reported 113 controlled experiments.

Although as many as 108 institutions from 19 countries were involved in conducting the experiments, a relatively low proportion of software engineering articles (1.9 percent) report controlled experiments, given that controlled experiments is the classical scientific method for identifying cause-effect relationships. One reason may be the large effort and resources needed to run well-designed experiments.

An important issue that pertains to all software engineering research is its relevance to the industry. For experiments, both the topics under investigation and how representative of an industrial setting an experiment is will influence industrial relevance. The two major areas investigated in the experiments were inspection techniques and object-oriented design techniques. This survey also gave some indications as to how realistic the experiments were relative to the subjects that took part, the tasks they performed, the types of applications on which these tasks were done, and the environment in which the subjects worked.

TABLE 18
Total Number of Articles Investigated

	Year										
Journal	**1993**	**1994**	**1995**	**1996**	**1997**	**1998**	**1999**	**2000**	**2001**	**2002**	**Total**
EMSE	-	-	-	10	24	14	17	19	24	16	124
ISESE	-	-	--	-	-	-	-	-	-	20	20
METRICS	15	11	-	17	18	32	31	-	30	23	177
JSS	87	78	76	74	82	91	95	112	101	90	886
TSE	85	74	77	65	52	83	55	68	62	76	687
ICSE	48	31	32	59	51	64	56	64	58	57	520
IST	67	69	62	69	76	80	87	83	78	74	745
SME	12	16	22	21	18	18	20	19	19	21	186
IEEE SW	50	56	45	51	52	48	59	60	55	56	532
TOSEM	13	12	10	13	12	13	13	14	11	14	125
IEEE Comp	70	76	74	83	91	79	78	73	81	75	780
SP&E	69	59	68	68	71	72	68	65	65	66	671
Total	**516**	**482**	**466**	**530**	**547**	**584**	**579**	**577**	**584**	**588**	**5453**

In total, 5,488 subjects participated in the experiments. The number of participants ranged from 4 to 266 with a mean value of 49. In total, 87 percent of the subjects were students, whereas only 9 percent were professionals. This indicates that one may question how representative the experimental results are for an industrial setting.

The same applies to the kind of application used in the experiments. In 75 percent, the applications were constructed for the purpose of the experiment or constituted student projects. Commercial applications were used in 14 percent of the experiments.

Threats to internal and external validity were addressed in, respectively, 63 percent and 69 percent of the experiments. Among the threats to internal validity, about 1/5 were not handled, 3/5 were reduced, and 1/5 were eliminated. This could either mean that the experiments all over had a high degree of internal validity or that the internal threats that were not reduced or eliminated were underreported. Threats to external validity regarding subject and task were discussed in more than half of the experiments, regarding environment in about 1/4 of the experiments and regarding treatment in only a few. Threats to internal validity regarding selection and instrumentation were most frequently reported.

A major finding of this survey is that the reporting is often vague and unsystematic and there is often a lack of consistent terminology. The community needs guidelines that provide significant support on how to deal with the methodological and practical complexity of conducting and reporting high-quality, preferably realistic, software engineering experiments. We recommend that researchers should accurately report the following: the type and number of subjects, including the mortality rate; context variables such as general software engineering experience and experience specific to the tasks of the experiments; how the subjects were recruited; the application areas and type of tasks; the duration of the tasks; and internal and external validity of the experiments, including being specific about the sample and target population of the experiment. A more uniform way of reporting experiments will help to improve the review of articles, replication of experiments, meta-analysis, and theory building.

TABLE 19
Number of Articles that Report Controlled Experiments

	Year										
Journal	**1993**	**1994**	**1995**	**1996**	**1997**	**1998**	**1999**	**2000**	**2001**	**2002**	**Total**
EMSE	-	-	-	2	6	5	1	5	1	2	22 (17.7% of 124)
ISESE	-	-	-	-	-	-	-	-	-	3	3 (15.0% of 20)
METRICS	0	0	-	2	0	4	0	-	3	1	10 (5.6% of 177)
JSS	1	1	1	4	0	4	5	6	1	1	24 (2.7% of 886)
TSE	2	1	2	0	2	1	1	3	3	2	17 (2.5% of 678)
ICSE	0	1	0	1	1	1	1	3	3	1	12 (2.3% of 520)
IST	0	0	0	1	2	2	0	0	3	0	8 (1.1% of 745)
SME	0	0	0	0	0	0	0	0	1	1	2 (1.1% of 186)
IEEE SW	0	0	0	0	0	0	0	3	1	0	4 (0.8% of 532)
TOSEM	0	0	0	0	0	1	0	0	0	0	1 (0.8% of 125)
IEEE comp	0	0	0	0	0	0	0	0	0	0	0 (0% of 780)
SP&E	0	0	0	0	0	0	0	0	0	0	0 (0% of 671)
Total	**3** **0.6%** **of 516**	**3** **0.6%** **of 482**	**3** **0.6%** **of 466**	**10** **1.9%** **of 530**	**11** **2.0%** **of 547**	**18** **3.1%** **of 584**	**8** **1.4%** **of 579**	**20** **3.5%** **of 577**	**16** **2.7%** **of 584**	**11** **1.9%** **of 588**	**103 (1.9% of 5453)**

Percentage of articles relative to total number of articles reported in Table 18.

APPENDIX

See Table 18 and Table 19.

ACKNOWLEDGMENTS

The authors are grateful to Barbara Kitchenham, Ray Welland, Erik Arisholm, Reidar Conradi, Tore Dybå, Magne Jørgensen, James Dzidek, and the anonymous reviewers for insight and feedback to several key issues covered in this survey. Thanks are due to Jørgen Busvold, Ingeborg Nygard, Ragnfrid Sjøberg, and Jørn Inge Vestgården for assistance in compiling data. Thanks to Chris Wright for proofreading the paper.

REFERENCES

[1] A. Abran and J.W. Moore, "SWEBOK—Guide to the Software Engineering Body of Knowledge," *2004 Version, IEEE CS Professional Practices Committee,* 2004.

[2] ACM Computing Classification, http://theory.lcs.mit.edu/jacm/CR/1991, 1995.

[3] E. Arisholm and D.I.K. Sjøberg, "Evaluating the Effect of a Delegated versus Centralized Control Style on the Maintainability of Object-Oriented Software," *IEEE Trans. Software Eng.,* vol. 30, no. 8, pp. 521-534, Aug. 2004.

[4] V.R. Basili, "The Experimental Paradigm in Software Engineering," *Experimental Eng. Issues: Critical Assessment and Future Directions, Proc. Int'l Workshop,* vol. 706, pp. 3-12, 1993.

[5] V.R. Basili, "The Role of Experimentation in Software Engineering: Past, Current, and Future," *Proc. 18th Int'l Conf. Software Eng.,* pp. 442-449, 1996.

[6] V.R. Basili, R.W. Selby, and D.H. Hutchens, "Experimentation in Software Engineering," *IEEE Trans. Software Eng.,* pp. 733-743, July 1986.

[7] V.R. Basili, F. Shull, and F. Lanubile, "Building Knowledge through Families of Experiments," *IEEE Trans. Software Eng.,* vol. 25, pp. 456-473, July/Aug. 1999.

[8] D.M. Berry and W.F. Tichy, "Response to 'Comments on Formal Methods Application: An Empirical Tale of Software Development,'" *IEEE Trans. Software Eng.,* vol. 29, no. 6, pp. 572-575, June 2003.

[9] D.T. Campbell and J.C. Stanley, "Experimental and Quasi-Experimental Designs for Research on Teaching," *Handbook of Research on Teaching,* N.L. Cage, ed., Chicago: Rand McNally, 1963.

[10] L.B. Christensen, *Experimental Methodology,* eighth ed. Boston: Pearson/Allyn & Bacon, 2001.

[11] T.D. Cook and D.T. Campbell, *Quasi-Experimentation. Design & Analysis Issues for Field Settings.* Houghton Mifflin, 1979.

[12] B. Curtis, "Measurement and Experimentation in Software Engineering," *Proc. IEEE,* vol. 68, no. 9, pp. 1144-1157, Sept. 1980.

[13] B. Curtis, "By the Way, Did Anyone Study Real Programmers?" *Empirical Studies of Programmers, Proc. First Workshop,* pp. 256-262, 1986.

[14] I.S. Deligiannis, M. Shepperd, S. Webster, and M. Roumeliotis, "A Review of Experimental Investigations into Object-Oriented Technology," *Empirical Software Eng.,* vol. 7, no. 3, pp. 193-231, Sept. 2002.

[15] A. Endres and D. Rombach, *A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories,* Fraunhofer IESE series on software engineering. Pearson Education Limited, 2003.

[16] N. Fenton, "How Effective Are Software Engineering Methods?" *J. Systems and Software,* vol. 22, no. 2, pp. 141-146, 1993.

[17] R. Ferber, "Editorial: Research by Convenience," *J. Consumer Research,* vol. 4, pp. 57-58, June 1977.

[18] R.L. Glass and T.Y. Chen, "An Assessment of Systems and Software Engineering Scholars and Institutions (1998-2002)," *J. Systems and Software,* vol. 68, no. 1, pp. 77-84, 2003.

[19] R.L. Glass, V. Ramesh, and I. Vessey, "An Analysis of Research in Computing Disciplines," *Comm. ACM,* vol. 47, no. 6, pp. 89-94, June 2004.

[20] R.L. Glass, I. Vessey, and V. Ramesh, "Research in Software Engineering: An Analysis of the Literature," *J. Information and Software Technology,* vol. 44, no. 8, pp. 491-506, June 2002.

[21] S. Greenland, J.M. Robins, and J. Pearl, "Confounding and Collapsibility in Causal Inference," *Statistical Science,* vol. 14, no. 1, pp. 29-46, 1999.

[22] W. Hayes, "Research Synthesis in Software Engineering: A Case for Meta- Analysis," *Proc. Sixth Int'l Symp. Software Metrics,* pp. 143-151, 2003.

[23] M. Höst, C. Wohlin, and T. Thelin, "Experimental Context Classification: Incentives and Experience of Subjects," *Proc. 27th Int'l Conf. Software Eng.,* pp. 470-478, 2005.

[24] IEEE Keyword Taxonomy, http://www.computer.org/mc/keywords/software.htm, 2002.

[25] M. Jørgensen, "A Review of Studies on Expert Estimation of Software Development Effort," *J. Systems and Software,* vol. 70, nos. 1,2, pp. 37-60, 2004.

[26] M. Jørgensen and D.I.K. Sjøberg, "Generalization and Theory Building in Software Engineering Research," *Empirical Assessment in Software Eng. Proc.,* pp. 29-36, 2004.

[27] M. Jørgensen, K.H. Teigen, and K. Moløkken, "Better Sure than Safe? Over-Confidence in Judgement Based Software Development Effort Prediction Intervals," *J. Systems and Software,* vol. 70, nos. 1,2, pp. 79-93, 2004.

[28] N. Juristo, A.M. Moreno, and S. Vegas, "Reviewing 25 Years of Testing Technique Experiments," *Empirical Software Eng.,* vol. 9, pp. 7-44, Mar. 2004.

[29] B.A. Kitchenham, "Procedures for Performing Systematic Reviews," Technical Report TR/SE-0401, Keele University, and Technical Report 0400011T.1, NICTA, 2004.

[30] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El-Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Trans. Software Eng.,* vol. 28, no. 8, pp. 721-734, Aug. 2002.

[31] R.M. Lindsay and A.S.C. Ehrenberg, "The Design of Replicated Studies," *The Am. Statistician,* vol. 47, pp. 217-228, Aug. 1993.

[32] C. Lott and D. Rombach, "Repeatable Software Engineering Experiments for Comparing Defect-Detection Techniques," *Empirical Software Eng.,* vol. 1, pp. 241-277, 1996.

[33] J.W. Lucas, "Theory-Testing, Generalization, and the Problem of External Validity," *Sociological Theory,* vol. 21, pp. 236-253, Sept. 2003.

[34] T.R. Lunsford and B.R. Lunsford, "The Research Sample, Part I: Sampling," *J. Prosthetics and Orthotics,* vol. 7, pp. 105-112, 1995.

[35] *Experimental Software Engineering Issues: Critical Assessment and Future Directions, Int'l Workshop Dagstuhl Castle (Germany), Sept. 14-18, 1992, Proc.,* H.D. Rombach, V.R. Basili, and R.W. Selby, eds. Springer Verlag, 1993.

[36] P. Runeson, "Using Students as Experimental Subjects—An Analysis of Graduate and Freshmen PSP Student Data," *Proc. Empirical Assessment in Software Eng.,* pp. 95-102, 2003.

[37] W.R. Shadish, T.D. Cook, and D.T. Campbell, *Experimental and Quasi-Experimental Designs for Generalized Causal Inference.* Houghton Mifflin, 2002.

[38] M. Shaw, "Writing Good Software Engineering Research Papers: Minitutorial," *Proc. 25th Int'l Conf. Software Eng.,* pp. 726-736, 2003.

[39] D.I.K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanović, E. Koren, and M. Vokáč, "Conducting Realistic Experiments in Software Engineering," *Proc. 18th Int'l Symp. Empirical Software Eng.,* pp. 17-26, 2002.

[40] D.I.K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanović, and M. Vokáč, "Challenges and Recommendations when Increasing the Realism of Controlled Software Engineering Experiments," *Empirical Methods and Studies in Software Eng.,* pp. 24-38, Springer Verlag, 2003.

[41] W.F. Tichy, "Should Computer Scientists Experiment More? 16 Excuses to Avoid Experimentation," *Computer,* vol. 31, no. 5, pp. 32-40, May 1998.

[42] W.F. Tichy, "Hints for Reviewing Empirical Work in Software Engineering," *Empirical Software Eng.,* vol. 5, no. 4, pp. 309-312, 2000.

[43] W.F. Tichy, P. Lukowicz, L. Prechelt, and E.A. Heinz, "Experimental Evaluation in Computer Science: A Quantitative Study," *J. Systems and Software,* vol. 28, no. 1, pp. 9-18, Jan. 1995.

[44] W.M.K Trochim, *The Research Methods Knowledge Base,* second ed., Cincinnati: Atomic Dog Publishing, 2001.

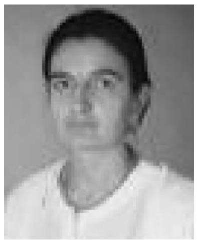
[45] R. Weber, "Editor's Comments," *MIS Quarterly,* vol. 27, no. 3, pp. iii-xii, Sept. 2003.
[46] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Eng.: An Introduction.* John Wiley & Sons Inc., 1999.
[47] R.K. Yin, *Case Study Research: Design and Methods.* Thousand Oaks, Calif.: Sage, 2003.
[48] E.A. Youngs, "Human Errors in Programming," *Int'l J. Man-Machine Studies,* vol. 6, no. 3, pp. 361-376, 1974.
[49] M.V. Zelkowitz and D. Wallace, "Experimental Validation in Software Engineering," *J. Information and Software Technology,* vol. 39, pp. 735-743, 1997.
[50] M.V. Zelkowitz and D. Wallace, "Experimental Models for Validating Technology," *Theory and Practice of Object Systems,* vol. 31, no. 5, pp. 23-31, May 1998.
[51] A. Zendler, "A Preliminary Software Engineering Theory as Investigated by Published Experiments," *Empirical Software Eng.,* vol. 6, no. 2, pp. 161-180, 2001.
[52] G.H. Zimney, *Method in Experimental Psychology.* New York: Ronald Press, 1961.

Dag I.K. Sjøberg received the MSc degree in computer science from the University of Oslo in 1987 and the PhD degree in computing science from the University of Glasgow in 1993. He has five years of industry experience as a consultant and group leader. He is now research director of the Department of Software Engineering, Simula Research Laboratory, and a professor of software engineering in the Department of Informatics, University of Oslo. Among his research interests are research methods in empirical software engineering, software process improvement, software effort estimation, and object-oriented analysis and design.

Jo E. Hannay received the MSc degree in computer science from the University of Oslo in 1995 and the PhD degree in type theory and logic from the University of Edinburgh in 2001. He has two years of industrial experience and now works as a postdoctoral fellow at Simula Research Laboratory. His interests revolve around research methodologies and the use of empirically-based theories in empirical software engineering.

Ove Hansen received the MSc degree in computer science from the University of Oslo in 2004. He now works as a part-time research assistant at Simula Research Laboratory.

Vigdis By Kampenes received the MSc degree in statistics from the University of Trondheim in 1993 and is now working on the PhD degree in software engineering at the University of Oslo and Simula Research Laboratory. She has eight years of experience as a statistician in the pharmaceutical industry and two years as a consultant in the IT industry. Her main research interest is research methods in empirical software engineering.



Amela Karahasanović received the MSc degree in computer science from the University of Sarajevo in 1989 and the PhD degree in computer science from the University of Oslo in 2002. She has nine years of industry experience in Bosnia-Herzegovina and Norway as a system developer and project manager. She is now a postdoctoral fellow at Simula Research Laboratory and an associate professor at the University of Oslo. Her research interests include research methods in empirical software engineering, visual languages, software comprehension, and object-oriented analysis and design.

Nils-Kristian Liborg received the MSc degree in computer science from the University of Oslo in 2004. He now works as a software developer at an international bank.

Anette C. Rekdal received the MSc degree in computer science from the University of Oslo in 2002. She then worked for two years as a research assistant at Simula Research Laboratory. She now works as a software developer at an IT company.