EXPERIMENTAL VALIDATION IN SOFTWARE ENGINEERING*

Marvin V. Zelkowitz
NIST/ITL
Gaithersburg, MD 20899
and Dept. of Computer Science
University of Maryland
College Park, MD 20742 USA

Dolores Wallace
NIST/ITL
Gaithersburg, MD 20899 USA

Abstract

Although experimentation is an accepted approach toward scientific validation in most scientific disciplines, it only recently has gained acceptance within the software development community. In this paper we discuss a 12 model classification scheme for performing experimentation within the software development domain. We evaluate over 600 published papers in the computer science literature and over one hundred papers from other scientific disciplines in order to determine: (1) how well the computer science community is succeeding at validating its theories, and (2) how computer science compares to other scientific disciplines.

Keywords: Data collection; Experimentation; Evaluation; Measurement

1 Experimentation

Experimentation and data collection are the tools of science for validating theories. The classical scientific method depends upon theory formation followed by experimentation and observation in order to provide a feedback loop to validate, modify and improve on the theory. Most of these concepts developed in the so-called "hard sciences" (e.g., physics, chemistry) hundreds of years ago and fields such as theoretical physics coexist along with experimental physics as opposing forces to develop new concepts and validate those concepts so developed.

Computer science is a relatively new field, with most academic departments forming during the late 1960s and 1970s. A strong experimental model of the field has not developed, at least as computer science folklore explains it.

 This paper was presented at the Empirical Assessment of Software Engineering Conference, Keele University, UK, March, 1997 and will be published in *Information and Software Technology* 39(11), November, 1997.

However, is this true? This question forms the basis for this paper. We study experimentation in computer science and have developed a 12-step model for experimental computer science. We classify over 600 papers in the computer science literature and show that our folk wisdom seems to be true – we do not do a good job in experimentation. We then survey several other scientific disciplines to see how our classification model and experimental paradigm fit other disciplines.

Software Engineering Experimentation. We have focused our concerns to the subdiscipline of software engineering, the study of tools and methods for producing quality software products. Experimentation has become the research focus of several groups. Data collection is central to the NASA/GSFC Software Engineering Laboratory [3], the concept behind the Data and Analysis Center for Software (DACS) located at Rome Laboratories and a crucial part of the upper levels of the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) [7]. However, how far has this technology spread across the entire subdiscipline? In fact, what does experimentation even mean in this domain?

Software engineering is concerned with techniques useful for the development of effective software programs, where "effective" depends upon specific problem domains. Effective software can mean software that either is low cost, reliable, rapidly developed, safe, or has some other relevant attribute. We make the assumption that to answer the question "Is this technique effective?" we need some measurement of the relevant attribute. Saying only that a technique is "good" conveys no real information. Instead, we need a measurement applied to each attribute so that we can say one technique is more or less effective than another.

For some attributes, this mapping from an effective attribute to a measurement scale is fairly straightforward. If effective means low cost, then cost of development is such a measure. For other applications of effectiveness, we have no good measurement scale. Safety is an example. What does it mean for one product to be safer than another? Security is another one of these nebulous attributes. What does it mean for one product to be more secure than another? Does it mean the time it takes to penetrate the software to bypass its security protection, how many data protection "items" it contains, or what level of information the program is allowed to process?

We summarize this then by saying that the basic purpose for experimentation is to determine whether methods used in accordance with some underlying theory during the development of a product results in software being as effective as necessary.

2 Data Collection Models

When we do an experiment, we are interested in the effect that a method or tool, called a *factor*, has on an attribute of interest. The running of an experiment with a specific assignment to the factors is called a *treatment*. Each agent that we are studying and collecting data on (e.g., programmer, team, source program module) is called a *subject*. The goal of an experiment is to collect enough data from a sufficient number of subjects, all adhering to the same treatment, in order to obtain a statistically significant result on the attribute of concern compared to some other treatment.

In developing an experiment to collect data on this attribute, Pfleeger mentions three aspects of data collection [8]:

1. *Replication* – The most important attribute of the scientific method is to be able to replicate the results of an experiment to permit other researchers to reproduce the findings. To ensure that this is so, we must not *confound* two effects. That is, we must make sure that unanticipated variables are not affecting our results.
2. *Local control* – Local control refers to the degree to which we can modify the treatment applied to each subject.
3. *Factorial design* – We have maximal local control by applying every possible treatment for each factor. Thus if there are three factors to evaluate, and each has two possible values, then we need to run six experiments, with subjects randomly chosen from among the factors.

There are two additional aspects, not mentioned by Pfleeger, to consider:

1. *Influence.* We need to know the impact that a given experimental design has on the results of that experiment. We will call this *influence* and classify methods as being *passive* or *active*. Passive methods view the artifacts of study as inorganic objects that can be studied with no effects on the object itself. Active methods are those which interact with the artifacts under study. Thus we introduce the possibility of contamination or modification of the results due to the very nature of our investigations. Active methods allow us to have some local control over the experiment, while passive methods provide for minimal perturbation of the results.

   In other scientific disciplines, a variety of experimental designs can be active or passive. Archaeology uses mostly passive designs, chemistry mostly active, while fields such as astronomy often uses a mixture of both.
2. *Temporal properties.* Experiments may be historical (e.g., archaeological) or current (e.g., monitoring a current project). There is certainly less control over the experimental design if the basic data was collected before the experiment began.

By looking at multiple examples of software engineering validation (e.g., the NASA Software Engineering Laboratory, the 600 published papers discussed later), we have developed a taxonomy for software engineering experimentation that describes 12 different experimental approaches. We are not claiming that this list of 12 is the ultimate list, but we have not seen any such list before that effectively categorizes multiple instances of experimental designs that are appropriate for our commmunity. We believe that this list is a good start for such an understanding of software engineering experimentation.

Experimentation is grouped into three categories:

1. An *observational* method will collect relevant data as a project develops. There is relatively little control over the development process.

2. An *historical* method collects data from projects that have already been completed. The data already exists; it is only necessary to analyze what has already been collected.

3. A *controlled* method provides for multiple instances of an observation in order to provide for statistical validity of the results.

2.1 Observational Methods

An *observational* method will collect relevant data as a project develops. There are four such methods: Project monitoring, Case study, Assertion, and Field study.

Project Monitoring. This is the collection and storage of data that occurs during project development. It is a passive model since the data will be whatever the project generates with no attempt to influence or redirect the development process or methods that are being used. The assumption is made that the data will be used for some immediate analysis. If an experimental design is constructed after the project is finished, then we would call this an historical lessons learned study.

This method lacks any experimental goals or consistency in the data that is collected. It is important, however, to collect this information so that a baseline can be established later, should the organization build a more complex experimentation process. Establishing baselines is crucial for later process improvement activities, such as applying Basili's Quality Improvement Paradigm (QIP) [1].

Case Study. A project is monitored and data collected over time. The important facet in this model is that the project is to be undertaken whether data is to be collected or not. With a relatively minimal addition to the costs to the project, valuable information can be obtained on the various attributes characterizing its development. (The NASA SEL claims that this adds only 1% to 2% to the cost of a project.)

This differs from the project monitoring method above in that data collection is focused on a specific goal for the project. A certain attribute is studied (e.g., reliability, cost) and data is collected to measure that attribute. Similar data is often collected from a class of projects so that a baseline is built that represents the organization's standard process for software development.

Assertion. There are many examples where the developer of a technology wishes to show that it is effective and becomes both the experimenter and the subject of the study. Sometimes this may be a preliminary test before a more formal validation of the effectiveness of the technology. But all too often, the experiment is a weak example favoring the proposed technology over alternatives. As skeptical scientists, we would have to view these as potentially biased since the goal is not to understand the difference between two treatments, but to show that one particular treatment (the newly developed technology) is superior. We will refer to such experiments as assertions.

The assertion category can be viewed as a value judgment by us on the effectiveness of an experiment performed by the developer. We had three alternatives for such experiments: (1) Consider it a true experiment, which we believed would not be appropriate in light of "good scientific principles;" (2) Consider it as no experiment at all, which we also believed was not appropriate since some validation, even if weak, was being performed on the technology; or (3) Reserve judgment on how to classify it by making it a separate category, which allows us to classify it as either of the first two categories as needed.

Field study. It is often desirable to compare several projects simultaneously. We view this as a cross between the project monitoring method, where any data is collected and the case study, where specific data is collected. An outside group will come and monitor the subject groups to collect the relevant information. Since the process is not modified, the data that can be collected is limited, but designed to achieve specific goals.

If the field study becomes very intrusive with a significant involvement of the development staff in the collection of the necessary data, then this method is a form of the replicated experiment to be described later.

2.2 Historical Methods

An *historical* method collects data from projects that have already been completed. There are four such methods: Literature search, Legacy data, Lessons learned, and Static analysis.

Literature Search. The literature search requires the investigator to analyze the results of papers and other documents that are publically available. This can be useful to confirm an existing hypothesis or to enhance the data collected on one project with data that has been previously published on similar projects (e.g., *meta-analysis* [6]).

This provides information across a broad range of industries, and access to such information is at a low cost. However, a major weakness with a literature search is *selection bias* or the tendency of researchers, authors, and journal editors to publish positive results. Contradictory results often are not reported, so a meta-analysis of previously published data may indicate an effect that is not really present if the full set of observable data was presented.

Legacy Data. A completed project leaves a legacy of products, called artifacts. These artifacts include the source program, specification document, design, and a test plan, as well as data collected during product development. We assume there is a fair amount of quantitative data available for analysis. When we do not have such quantitative data, we call the analysis a lessons learned study (described later). We will also consider the special case of looking at source code and specification documents alone under the separate category of static analysis.

Legacy data is a low cost form of experimentation. It can be called a form of *software archaeology*

as we examine existing files trying to determine trends. *Data mining* is another term often used for parts of this work as we try to determine relationships buried in the collected data.

Lessons-learned. Lessons-learned documents are often produced after a large industrial project is completed, whether data is collected or not. A study of these documents often reveals qualitative aspects which can be used to improve future developments. If project personnel are still available, it is possible to interview them to obtain trends in looking at the effects of methods.

Static Analysis. We can often obtain needed information by looking at the completed product. We analyze the structure of the product to determine characteristics about it. Software complexity and data flow research fit under this model. For example, since we do not fully understand what the effective measurements are, the assumption is made that products with a lower complexity or simple data flow will be more effective. We examine the product to learn if its complexity value is lower because of the development method used.

2.3 Controlled Methods

A *controlled* method provides for multiple instances of an observation in order to provide for statistical validity of the results. This is the more classical method of experimental design in other scientific disciplines. We consider four such methods: Replicated, Synthetic environment, Dynamic analysis, and Simulation.

Replicated Experiment. In a replicated experiment several projects (e.g., subjects) are staffed to perform a task using alternative treatments. Ideally, several projects are staffed to perform each of the possible treatments. Control variables are set (e.g., duration, staff level, methods used) and statistical validity can be more easily established than the large case study previously mentioned. On the other hand, the risk of perturbing the experimental results is great since the subjects know they are part of an experiment and not part of a true development.

If there are enough replications, statistical validity of the method under study may be established. Since this is usually part of an industrial setting, the transfer of this technology to industry should be apparent, and the risk of using the results of this study should be lessened. However, the cost of such replications is great, limiting their usefulness.

Synthetic Environment Experiments. In software development, projects are usually large and the staffing of multiple projects (e.g., the replicated experiment) in a realistic setting is usually prohibitively expensive. For this reason, most software engineering replications are performed in a smaller artifical setting, which only approximates the environment of the larger projects. Because we believe that such experiments are performed in a different environment from the industrial setting of a true development, we call these synthetic environment experiments.

Such experiments often appear as human factors studies investigating some aspect in system design or use. Typically, a large group of individuals (e.g., students or industrial programmers) work at some task for several hours, leading to data being collected on this task. A relatively small objective is identified and all variables are fixed except for the control method being modified. Personnel are often randomized from a homogeneous pool of subjects, duration of the experiment is fixed, and as many variables as possible are monitored.

Like the replicated experiment, it can be used to obtain a high degree of statistical validity, but even more than the replicated experiment, the artificial experimental setting of the experiment (e.g., classroom or laboratory) may perturb the results to make their applicability to an industrial setting very suspect.

Because the objectives of such experiments are often limited, the relevance of transferring the results of such experiments to industry may be limited. In this case it is not clear that the experimental design relates to the environment that already exists in industry. So we may end up with valid statistics of an experimental setup for a method that may not be applicable.

Dynamic Analysis. The controlled methods we have so far discussed generally evaluate the development process. We can also look at controlled methods that execute the product. We call these dynamic analysis methods.

The given product is either modified or executed under carefully controlled situations in order to extract information on using the product. Techniques that employ scripts of specific scenarios or which modify the source program of the product itself in order to be able to extract information while the program executes are both examples of this method. *Benchmarking* occurs when a common script is used to evaluate several products that have the same functionality in order to compare the performance of each.

Simulation. We can evaluate a technology by executing the product using a model of the real environment. We hypothesize, or predict, how the real environment will react to the new technology. If we can model the behavior of the environment for certain variables, we often can ignore other harder-to-obtain variables and obtain results more readily using a simulated environment rather than real data.

By ignoring extraneous variables, a simulation is often easier, faster, and less expensive to run than the full product in the real environment (i.e., the dynamic analysis method).

2.4 Alternative Classification Models

Several other frameworks for experimentation have been proposed. Basili [2] calls an experiment *in vivo*, at a development location, or *in vitro*, in an isolated controlled setting (e.g., in a laboratory). A project may involve one team of developers or multiple teams, and an experiment may involve one project or multiple projects. This permits 8 different experiment classifications. Our case study,

for example, would be an in vivo experiment involving one team and one project. The synthetic study, on the other hand, is often a multiple team blocked (multiple individuals in most cases) in vitro study involving one project.

Kitchenham [5] considers experiments as: *quantitative experiments* to identify measurable benefits of using a method or tool; a *qualitative experiments* to assess the features provided by a method or tool (e.g., the usability and effectiveness of a required feature, training requirements); and *benchmarking* where a number of standard tests are run against alternative technologies in order to assess their relative performance.

We believe that both of the above taxonomies are subsets of the 12 we have given here and do not include all 12 of the experimental models we have presented.

3 Model Validation

Given our taxonomy, we evaluated the software engineering literature and did a brief overview of the literature from other scientific disciplines in order to determine: (1) which methods were used most frequently, and (2) how well computer science compares to other disciplines.

3.1 Software Engineering Literature

To test whether the classification presented here reflects the software engineering community's idea of experimental design and data collection, we evaluated software engineering publications covering three different years: 1985, 1990, and 1995. We looked at each issue of *IEEE Transactions on Software Engineering* (a research journal), *IEEE Software* (a magazine which discusses current practices in software engineering), and the proceedings from that year's International Conference on Software Engineering (ICSE). We classified each paper according to our taxonomy according to how the authors validated their claims in the paper. For completeness we needed the following two additional classifications:

1. *Not applicable.* Some papers did not address some new technology, so the concept of data collection did not apply (e.g., a paper summarizing a recent conference or workshop).
2. *No experiment.* Some papers describing a new technology contained no experimental validation in it. We do not put a value judgment on this and *no experiment* is not the same as *bad experimental validation.* For example, a paper that describes a new theory may be quite important and useful to the field. It would be up to the next generation of researchers to implement and evaluate the effectiveness of the proposed technology.

According to Glass, a research paper typically contains four sections [4]:

1. An informational phase reflecting the context of the proposed technology,

2. A propositional phase stating the hypothesis for the new technology,

3. An analytical phase analyzing the hypothesis and proposing a solution, and

4. An evaluative phase demonstrating the validity of the proposed solution.

Glass observed that most papers in his study contained some form of the first three sections, and the fourth evaluative phase was often missing.

Tichy [9] performed a comprehensive study of 400 published papers in the computer science literature (mostly ACM journals and conferences) and arrived at a similar conclusion. He classified all papers into formal theory, design and modeling, empirical work, hypothesis testing, and other. His major observation was that about half of the design and modeling papers did not include experimental validation, whereas only 10% to 15% of papers in other engineering disciplines had no such validation.

In our survey, we were most interested in the data collection methods employed by the authors of the paper in order to determine comprehensiveness of our classification scheme. Therefore, we tried to carefully distinguish between Glass' analytical and evaluative phase in order to carefully distinguish between demonstration of concept (which may involve some measurements as a "proof of concept," but not a full validation of the method) and a true attempt at validation of the results. Therefore, as in the Tichy study, a demonstration of a technology via an example was considered part of the analytical phase. The paper had to go beyond that demonstration to show that there were some conclusions about the effectiveness of the technology before we considered that the paper had an evaluative part.

Table 1 (Summary totals) summarizes the 612 papers from 1985, 1990, and 1995 that we classified.[1] (We did not include the 50 "not applicable" papers in computing the percentages in this table.) The most prevalent validation mechanisms appear to be lessons learned and case studies, each at a level of about 10%. Assertions were close to one-third of the papers. Simulation was used in about 5% of the papers, while the remaining techniques were each used in about 1% to 3% in the papers.

About one-third of the papers had no experimental validation; however, the percentages dropped from 36.4% in 1985 to 29.2% in 1990 to only 19.4% in 1995. Improvement in this important category seems to be occurring.

From Tichy's classification, many "empirical work" papers really are the result of an experiment to test a theoretical hypothesis, so it may not be fair to ignore those papers from the set of design and modeling papers. If we assume the 25 empirical work papers in Tichy's study all have evaluations in them, then the percent of design and modeling papers with no validation drops from 50% to about 40% in his study. (These numbers are approximate, since we don't have the details of his raw data.) This number is consistent with our results.

We did not try to classify our database of papers into subject matter, so our results are not strictly comparable with Tichy's. However, by combining the no experimental validation papers

[1] The complete breakdown by magazine is given in the appendix.

Method	1985	%	1990	%	1995	%	Total	%
Not applicable	15	–	22	–	13	–	50	–
No experimentation	83	36.4	57	29.2	27	19.4	167	29.7
Replicated	1	0.4	1	0.5	4	2.9	6	1.1
Synthetic	5	2.2	5	2.6	2	1.4	12	2.1
Dynamic analysis	0		3	1.5	4	2.9	7	1.2
Simulation	12	5.3	11	5.6	8	5.8	31	5.5
Project monitoring	0		1	0.5	0		1	0.2
Case study	19	8.3	19	9.7	20	14.4	58	10.3
Assertion	79	34.6	73	37.4	40	28.8	192	34.2
Field study	2	0.9	1	0.5	4	2.9	7	1.2
Literature search	5	2.2	7	3.6	5	3.6	17	3.0
Legacy data	4	1.8	4	2.1	3	2.2	11	1.9
Lessons learned	16	7.0	13	6.7	20	14.4	49	8.7
Static analysis	2	0.9	0		2	1.4	4	0.7
Yearly totals	243		217		152		612	

Table 1: Classification of software engineering papers.

with the weak form of assertion validation, we found that almost two-thirds of the papers did not have strong validation of their reported claims. However a claim that 64% of the papers had no validation is too strong a statement to make, since the assertion papers did include some form of quantitative analysis of the effects of their technology.

3.2 Comparison with Other Sciences

In order to provide a proper perspective, we looked at several different journals in other scientific disciplines. Because of the thousands of available journals and the limited resources of the authors to perform this survey, the following is necessarily incomplete and only represents a small random sampling of the possibilities. The method we chose was to evaluate several consecutive issues of each journal published between 1992 and 1996. We wanted about 20 papers from each journal. (We soon discovered that classifying papers in fields in which we were not experts took considerably more time than those in our more familiar software engineering domain.) The results of this sampling are given in Table 2.

Although our sample size is very small and we make no claims that these are representative of each respective field, some trends seem apparent:

- Not too surprizingly, each field seems to have a characteristic pattern of canonical data collection methods. Physics journals tend to report on measurement devices or reactions, so validation tends to be in repeated trials of the device to confirm behavior (e.g., techniques like dynamic analysis and simulation). Psychology is concerned about human behavior, so

Method	J 1	J 2	J 3	J 4	J 5	J 6	Ttl	%
Not applicable (Number)		2	5			1	8	–
No experimentation	16%	58%	7%	21%	6%	31%	26	20
Replicated		5%	4%	4%	12%		5	–
Synthetic			4%	11%	29%		9	–
Dynamic analysis	32%	5%	19%	11%			17	–
Simulation			15%	32%			13	–
Project monitoring								
Case study	40%	16%	41%		6%	8%	26	–
Assertion	8%	4%	11%			8%	7	5
Field study				4%	18%		4	–
Literature search	4%	11%	7%	7%	24%	23%	14	–
Legacy data					6%	23%	4	–
Lessons learned		5%				8%	2	–
Static analysis								
Paper count (Number)	25	21	32	28	17	14	137	—

J 1 – Measurement Science and Technology, Jan-Feb, 1994 (Devices to perform measurements).
J 2 – American Journal of Physics, Jan-Feb, 1996 (Theory and application of physical theories).
J 3 – Journal of Research of NIST, Sept. 1991-Aug., 1992 (Measurement and standardization issues).
J 4 – Management Science, Jan.-Mar., 1992 (Queueing theory and scheduling).
J 5 – Behavior Therapy, Winter-Summer, 1996 (Clinical therapies.)
J 6 – Journal of Anthropological Research, Winter-Summer, 1996 (Human cultures.)

Table 2: Classification from other sciences.

its journals tend to report on longitudinal studies of subjects who had different treatments (e.g., replicated and synthetic experiments). Anthropological papers used mostly passive techniques on historical data (e.g., legacy data). Because of these differing characteristics, we did not believe a summary "ranking" of techniques across all 6 surveyed journals made any sense, so we omitted those numbers in Table 2, except for the no experimentation and assertion entries, which we discuss below.

Anthropology and psychology also tend to use literature search as a significant method for validation, much more than in the other journals. The literature search seems to be in the range of 4% to 11% across the physics journals we looked at. This may indicate that other disciplines are more willing to publish papers that confirm previously published results. We believe (although this is only known anecdotally) that the computer science community frowns on publishing papers if a previous study by another author has been published on a given topic. In our software engineering sample, the per cent of literature survey grew from 2.1% to 3.3% to 3.5% from 1985 to 1995, so perhaps the situation is improving in our own field.

	1985	1990	1995	Total	%
ICSE conferences					
Theory	3	1	3	7	20.6
No validation	13	7	7	27	79.4
IEEE Software Magazine					
Theory	1	0	0	1	4.5
No validation	10	8	3	21	95.5
Transactions on Software Engineering					
Theory	18	19	7	44	39.6
No validation	38	22	7	67	60.4
Summary totals					
Theory	22	20	10	52	31.1
No validation	61	37	17	115	68.9
No Experimentation papers	83	53	27	167	—

Table 3: No experimentation papers.

- In the physics journals, except for *American Journal of Physics*, the per cent of no experimentation papers was much lower than in software engineering (7% to 21% compared to 30% in software engineering.) However, the trends in software engineering look favorable, with the per cent of no experimentation dropping from 36% to 29% to 19% from 1985 to 1995.

  In looking at the 11 "No experimentation" papers in physics, 9 were theoretical papers developing new theories based upon some formal model that were not applicable to experimentation. Only two of the papers (11%) appeared to lack quantitative data where such data could have been presented.

  We noticed that anthropology also had a high rate of no experimentation, coupled with a high percentage of literature searches. Perhaps the high cost of field trips to available anthropological sites to obtain original data has allowed the field to develop in this manner.

  We checked for this same phenomenon as seen in physics in our software engineering sample by reevaluating all the papers originally classified as "no experiment" (Table 3). Of these 167 papers, about one-third (31.1%) could be called theoretical papers. (The papers had to have a formal inference component in order to be called a theoretical paper. A paper that was simply a description of a new process, language, or technique with no formal component to prove that the concept had the desired properties claimed by the authors was still considered a "no experiment" paper.)

  From Table 3 we also see, not unsurprizingly, that different publications have different classes of papers. *IEEE Software*, as a general purpose magazine, has few theoretical papers, and the ICSE conference has fewer theoretical papers than the archival journal (*Trans. on Software Engineering*).

  By looking only at the 340 papers from the archival journal, we get the complete classification given in Table 4. According to this table, about two-thirds of the papers (223 out of 334, not counting the not applicable papers) were classified according to our 12 method taxonomy,

	Count	%
Not applicable	6	–
No experimentation	67	20.1
Theory	44	13.2
Some experimentation	223	66.8
Total	340	100.0

Table 4: Transactions on Software Engineering paper classification.

about 13% were theoretical papers, and about 20% lacked any validation. This twenty percent figure is not out of line with the other disciplines we looked at.

- Although the 20% "no experimentation" figure for *TSE* papers looks fairly good, about 34% of our sample of classified papers were assertions (including 118 of the 223 classified *TSE* papers). Assertion papers are relatively rare across the other journals we looked at (from 0% to 11% in our sample).

  If we subtract the 9 theoretical papers in the physics journal from the total of 28 no experimentation papers and 7 assertion papers, the total of 26 papers with no or little experimentation (20%) is similar to the 15% found by Tichy in his study of other disciplines. The corrsponding numbers for software engineering are 20% no experimentation with 34.2% assertions, or just over 50% of the sample of software engineering papers should have done a better job of validating their claims.

- Many journals use a standard format in the organization of technical papers that includes presentation of the hypothesis, development of the concept, and experimental validation. Most papers included a section called "Experiment." The *Behavior Therapy* papers all had sections titled "Method," "Procedures," and "Results," making the classification of such papers quite easy. Such rigid formatting is certainly not true in computer science.

4 Conclusions

In this paper we have developed a 12 step taxonomy for classifying experimental validation in the scientific literature. We classified 612 papers in the software engineering literature and found that the method appears to work quite well. In addition, we confirmed some of the observations of Glass and Tichy that the software engineering community does not do a good job in validating its claims.

We observed that 20% of the papers in the *IEEE Transactions on Software Engineering* have no validation component (either experimental or theoretical). This number is comparable to the 15% to 20% percent observed in other scientific disciplines. However, about one-third of the software engineering papers had a weak form of experimentation (assertions) where the comparable figure for other fields was more like 5% to 10%.

We were encouraged by these findings that in about 80% of the papers, the authors realized that a proper scientific paper needs to validate its claims. This goes against the usual folklore that software engineering is simply ad hoc system building with no thoughts to the underlying science of the field. However, the real weakness in our field seems to be in our assertion category. These are papers where the authors realize they need to validate their claims, but we believe (albeit subjectively) that their validation was generally insufficient. We believe that this large assertion category represents a lack of attention on the part of authors. If we, as a discipline, start to require more precise validation, this number could be brought down fairly quickly.

We also reviewed over 100 papers from several other scientific disciplines. Each field seems to have characteristic methods appropriate for its problem domain. We seem to confirm the contention that other fields do a better job of validating scientific claims. The assertion classification is infrequently used and literature search validation is more prevalent than in software engineering. We must restate, of course, that our sampling of the other journals was by an informal random process and subject to change if we perform as detailed a study of those disciplines as we believe we have done in software engineering.

Software engineering seems unique among the disciplines we studied. We have process research looking at groups of individuals, much like the longitudinal studies of psychology, we have post mortems to understand a completed project, much like anthropology, and we develop tools to perform actions, much like physics. But we often are developing a "first of its kind" device or process, so comparable data is not easily obtainable. We would expect a characteristic software engineering profile to be more varied than the other fields we looked at.

Our next step will be to refine these 12 approaches more precisely and look more closely at what data each of the techniques can provide to the experimentor. Our goal is to provide a means that can be used to choose an experimental paradigm that can be used to collect a specific type of evaluation data.

We do not wish to end this paper with the claim that software engineering is an inferior and sloppy science. Software engineering is also a comparatively new field. Unlike physics, there do not exist older devices from which comparative data can be easily obtained. Therefore, it often is hard to obtain good quantitative data from which to make an evaluation. However, we certainly can do better than we are currently doing. It is encouraging that the trends from 1985 to 1990 to 1995 are moving in the right direction.

5 Acknowledgment

We acknowledge the help of Dale Walters of NIST who helped to classify the 600 papers used to validate the classification model described in this paper.

References

[1] Basili V. R. and H. D. Rombach, The TAME project: Towards improvement-oriented software environments, *IEEE Trans. on Soft. Eng.* 14, 6 (1988) 758-773.

[2] Basili, V. R., The Role of Experimentation: Past, Present, Future, (Keynote presentation), 18^{th} International Conference on Software Engineering, Berlin, Germany, March, 1996.

[3] Basili V., M. Zelkowitz, F. McGarry, J. Page, S. Waligora, and R. Pajerski, SEL's software process-improvement program, *IEEE Software* 12, 6 (1995) 83-87.

[4] Glass R. L., A structure-based critique of contemporary computing research, *J. of Systems and Software*, Vol. 28, No. 1 (1995), 3-7.

[5] Kitchenham B. A., Evaluating software engineering methods and tool, *ACM SIGSOFT Software Engineering Notes*, (January, 1996) 11-15.

[6] National Research Council, *Combining Information: Statistical Issues and Opportunities for Research*, Panel on Statistical Issues and Opportunities for Research in the Combination of Information, National Academy Press, Washington, DC, 1992.

[7] Paulk, M. C., B. Curtis, M. B. Chrissis and C. V. Weber, Capability Maturity Model for Software, Version 1.1, *IEEE Software*, Vol. 10, No. 4, (1993) 18-27.

[8] Pfleeger S. L., Experimental design and analysis in software engineering, *Annals of Software Engineering* 1 (1995) 219-253.

[9] Tichy W. F., P. Lukowicz, L. Prechelt, and E. A. Heinz, Experimental evaluation in computer science: A quantitative study, *J. of Systems and Software* Vol. 28, No. 1 (1995) 9-18.

[10] Zelkowitz M. V. and D. Wallace, Experimental models for software diagnosis, Natl. Inst. of Stnds. and Tech., NISTIR 5889, September, 1996.

Appendix

The following tables are the complete set of data for the 612 papers that we analyzed. The following tables are summarized by Table 1.

Method	**1985**	%	**1990**	%	**1995**	%	**Total**	%
Not applicable	6	–	4	–	5	–	15	–
No experimentation	16	32.0	8	25.8	10	37.0	34	31.5
Replicated	1	2.0	0		1	3.7	2	1.9
Synthetic	3	6.0	0		0		3	2.8
Dynamic analysis	0		0		0		0	
Simulation	2	4.0	0		1	3.7	3	2.8
Project monitoring	0		0		0		0	
Case study	5	10.0	7	22.6	4	14.8	16	14.8
Assertion	12	24.0	12	38.7	4	14.8	28	25.9
Field study	1	2.0	0		1	3.7	2	1.9
Literature search	1	2.0	1	3.2	0		2	1.9
Legacy data	1	2.0	2	6.5	1	3.7	4	3.7
Lessons learned	7	14.0	1	3.2	5	18.5	13	12.0
Static analysis	1	2.0	0		0		1	0.9

Table 5: International Conference on Software Engineering

Method	**1985**	%	**1990**	%	**1995**	%	**Total**	%
Not applicable	6	–	16	–	7	–	29	–
No experimentation	11	32.4	8	18.2	3	8.3	22	19.3
Replicated	0		0		0		0	
Synthetic	1	2.9	1	2.8	0		2	1.8
Dynamic analysis	0		0		0		0	
Simulation	0		0		1	2.8	1	0.9
Project monitoring	0		1	2.8	0		1	0.9
Case study	2	5.9	6	13.6	6	16.7	14	12.3
Assertion	13	38.2	19	43.2	14	38.9	46	40.4
Field study	0		0		1	2.8	1	0.9
Literature search	1	2.9	5	11.4	3	8.3	9	7.9
Legacy data	1	2.9	0		1	2.8	2	1.8
Lessons learned	5	14.7	4	9.1	7	19.4	16	14.0
Static analysis	0		0		0		0	

Table 6: IEEE Software Magazine.

Method	**1985**	%	**1990**	%	**1995**	%	**Total**	%
Not applicable	3	–	2	–	1	–	6	–
No experimentation	56	38.9	41	34.2	14	18.4	111	32.6
Replicated	0		1	0.8	3	3.9	4	1.2
Synthetic	1	0.7	4	3.3	2	2.6	7	2.1
Dynamic analysis	0		3	2.5	4	5.3	7	2.1
Simulation	10	6.9	11	9.2	6	7.9	27	7.9
Project monitoring	0		0		0		0	
Case study	12	8.3	6	5	10	13.2	28	8.2
Assertion	54	37.5	42	35.0	22	28.9	118	34.7
Field study	1	0.7	1	0.8	2	2.6	4	1.2
Literature search	3	2.1	1	0.8	2	2.6	6	1.8
Legacy data	2	1.4	2	1.7	1	1.3	5	1.5
Lessons learned	4	2.7	8	6.6	8	10.5	20	5.9
Static analysis	1	0.7	0		2	2.6	3	0.9

Table 7: IEEE Transactions on Software Engineering.