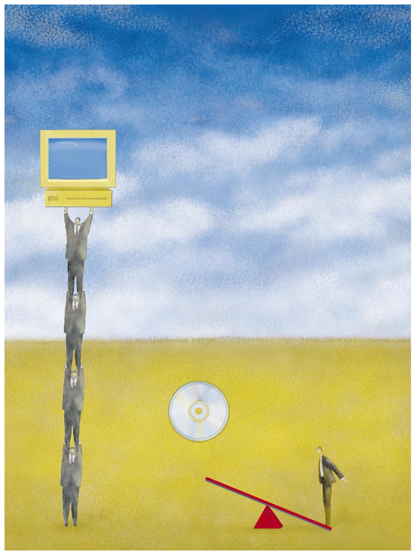
Experimental Models for Validating Technology



Experimentation helps determine the effectiveness of proposed theories and methods. But computer science has not developed a concise taxonomy of methods for demonstrating the validity of new techniques.

Marvin V. Zelkowitz
University of Maryland

Dolores R. Wallace
National Institute of Standards and Technology

Effective software can mean software that is low cost, reliable, rapidly developed, safe, or has some other relevant attribute. To determine whether a particular technique is effective, we need a way to measure it. Merely saying that a technique is effective conveys no real information. Instead, we need measurements applied to each software attribute so we can say one technique is more or less effective than another.

For some techniques, mapping from an effective attribute to a measurement scale is fairly straightforward. If "effective" means "low cost," then cost of development would be such a measurement. For other attributes—such as reliability, safety, and security—measurements are not so easily taken. Measurements like the number of failures per day, errors found during development, or MTBF (mean time between failures) indicate reliability in hardware domains. For software, however, a count of the number of errors found during testing does not by itself indicate whether there are errors remaining to be found. Only refined experimentation can help determine that.

Experimentation is a crucial part of attribute evaluation and can help determine whether methods used in accordance with some theory during product development will result in software being as effective as necessary. Should we modify the underlying theory upon which the technique is based? What predictions can we make about future developments based upon using these techniques?

Experimentation is one of those terms that is frequently used incorrectly in the computer science community. Researchers write papers that explain some new technology; then they perform "experiments" to show how effective the technology is. In most cases, the creator of the technology both implements the technology and shows that it works. Very rarely does such experimentation involve any collection of data to show that the technology adheres to some underlying model or theory of software development or that the software is effective.

Without a confirming experiment, why should industry select a new method or tool? On what basis should researchers enhance a language (or extend a method) and develop supporting tools? In a scientific discipline, we need to do more than simply say, "I tried it, and I like it."

HOW DO WE EXPERIMENT?

When we think of an experiment, we often think of a roomful of subjects, each being asked to perform some task. The task is usually followed by data collection and then analysis. This is certainly one type of experimen-

Table 1. Summary of software engineering validation models.

Validation method	Category	Description	Weakness
Project monitoring	Observational	Collect development data	No specific goals
Case study	Observational	Monitor project in depth	Poor controls for later replication
Assertion	Observational	Use ad hoc validation techniques	Insufficient validation
Field study	Observational	Monitor multiple projects	Treatments differ across projects
Literature search	Historical	Examine previously published studies	Selection bias; treatments differ
Legacy	Historical	Examine data from completed projects	Cannot constrain factors; data limited
Lessons learned	Historical	Examine qualitative data from completed projects	No quantitative data; cannot constrain factors
Static analysis	Historical	Examine structure of developed product	Not related to development method
Replicated	Controlled	Develop multiple versions of product	Very expensive; Hawthorne effect
Synthetic	Controlled	Replicate one factor in laboratory setting	Scaling up; interactions among multiple factors
Dynamic analysis	Controlled	Execute developed product for performance	Not related to development method
Simulation	Controlled	Execute product with artificial data	Data may not represent reality; not related to development method

tation. But there are other approaches as well, each of which can be grouped into four general categories:[1]

- *Scientific method.* Scientists develop a theory to explain a phenomenon; they propose a hypothesis and then test alternative variations of the hypothesis. As they do so, they collect data to verify or refute the claims of the hypothesis.
- *Engineering method.* Engineers develop and test a solution to a hypothesis. Based upon the results of the test, they improve the solution until it requires no further improvement.
- *Empirical method.* A statistical method is proposed as a means to validate a given hypothesis. Unlike the scientific method, there may not be a formal model or theory describing the hypothesis. Data is collected to verify the hypothesis.
- *Analytical method.* A formal theory is developed, and results derived from that theory can be compared with empirical observations.

The common theme of these methods is the collection of data on either the development process or the product itself. When we do an experiment using the scientific method, we are interested in the effect that a method or tool, called a *factor*, has on an attribute of interest. Running an experiment with a specific assignment of an effect the factor should achieve is called a *treatment*. Each agent that we study and collect data on—such as programmer, team, or source program module—is called a *subject* or an *experimental unit*. 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 an attribute, we have to be concerned with several aspects of data collection:[2]

- *Replication.* We must be able to replicate the results of an experiment to permit other researchers to reproduce the findings. We must not confound two effects. We must make sure that unanticipated variables are not affecting our results. If we cannot get a homogeneous sample of subjects for all treatments, we counteract this confounding effect by *randomizing* the factors that we are not concerned about.
- *Local control.* Local control refers to the degree to which we can modify the treatment applied to each subject. For example, we usually have little control over the treatment in a case study. Local control is a major problem in computer science research, since many of the treatments incur significant costs or expenditures of time. In a *blocking* experiment, we assume each subject of a treatment group comes from a homogeneous population. If we randomly select subjects from a population of students, we say that we have a blocked experiment of students. In a *factorial design*, we apply every possible treatment for each factor. Thus, if there are three factors to evaluate, and each has three possible values, then we need to run nine experiments with subjects randomly chosen from among the blocked factors.

With software development, there are two additional aspects to consider:

- *Influence.* In developing experiments involving large, complex, and expensive methods, such as

Strength
Provides baseline for future; Inexpensive
Can constrain one factor at low cost
Serves as a basis for future experiments
Inexpensive form of replication
Large available database; Inexpensive
Combines multiple studies; Inexpensive
Determine trends; Inexpensive
Can be automated; Applies to tools
Can control factors for all treatments
Can control individual factors; moderate cost
Can be automated; Applies to tools
Can be automated; Applies to tools; Evaluation in safe environment

software development, we need to know the impact—that is, the influence—that a given experimental design has on the results of an experiment. We classify the various methods as *passive* (viewing the artifacts of study as inorganic objects that can be studied with no effects on the objects themselves) or *active* (interacting with the artifacts under study, often affecting the behavior of the objects, as in the case of the Hawthorne effect defined).

- *Temporal properties.* Data collection may be historical (for example, archaeological) or current (for example, monitoring a current project). Historical data will certainly be passive, but may be missing just the information we need to come to a conclusion.

These categories and aspects of experimentation apply to science in general, but for effective experimentation in software engineering, we need to implement approaches specific to the characteristics of software. In the remainder of this article, we describe several such approaches and the results of a study examining how these approaches have been used.

VALIDATION MODELS

By looking at multiple examples of technology validation, we 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 that effectively categorizes multiple instances of experimental designs that are appropriate for our community. This list is a good place to start for such an understanding of software engineering experimentation. Table 1 summarizes these 12 models, and the following sections describe them in greater detail.

The various data collection methods shown in Table 1 can be grouped into three broad categories:

- *observational*,
- *historical*, and
- *controlled*.

An observational method collects relevant data as a project develops. There is relatively little control over the development process other than through using the new technology that is being studied. A historical method collects data from projects that have already been completed. The data already exist; it is only necessary to analyze what has already been collected. A controlled method provides for multiple instances of an observation for statistical validity of the results. This method is the classical method of experimental design in other scientific disciplines.

Observational methods

An *observational* method generally collects relevant data as a project develops. There are four types: project monitoring, case study, assertion, and field study.

Project monitoring. Project monitoring represents the lowest level of experimentation and measurement. It is the collection and storage of data that occurs during project development. It is a passive model, since the available data will be whatever the project generates; the researchers do not attempt to influence or redirect the development process or methods being used. Researchers assume the data will be used for some immediate analysis. If an experimental design is constructed after the project is finished, then we call this a historical lessons-learned study.

A problem with project monitoring is the difficulty in retrieving information later. A 1982 survey[3] found that although researchers often collected project information, such information is owned by the project manager and might not be available for future projects. The solution to this problem requires some minimal coordination among the various development activities in an organization. While this method lacks any experimental goals or consistency in the collected data, collecting this information enables researchers to establish a baseline, such as Victor Basili's Quality Improvement Paradigm (QIP).[4]

Case study. In a case study, researchers monitor a project and collect data over time. With a relatively minimal addition to project costs, valuable information can be obtained on the various attributes characterizing its development. Unlike the project monitoring method, data collection for a case study is derived from a specific goal for the project. Researchers monitor a certain attribute—such as reli-

All too often the experiment is a weak example favoring the proposed technology over alternatives. Skeptical scientists would have to view these experiments as potentially biased.

ability or cost—and collect data to measure that attribute. Researchers often collect similar data from a class of projects to build a baseline; then they use the baseline to represent the organization's standard process for software development.

A case study is an active method because of the influence humans may have on the development process itself. The very nature of filling out a form—with details like hours worked or errors found—might not by itself be intrusive, but it may have the side effect of having the staff think about and react to certain issues that emerge in the study.

The strength of this method is that the development is going to happen regardless of the needs to collect experimental data, so the only additional cost is the cost of monitoring the development and collecting this data. If an organization is attuned to the needs of experimentation and data collection, researchers can amass data from many projects over a short period of time.

The weakness of this method is that each development is relatively unique, so it is not always possible to compare one development profile with another. Determining trends and statistical validity is often difficult. Furthermore, because case studies are often large commercial developments, the needs of today's customer often dominate over the desire to learn how to improve the process later. The practicality of completing a project on time—within budget and with appropriate reliability—may mean that experimental goals must be sacrificed. Experimentation may be a risk that management is not willing to undertake.

Assertion. There are many examples of developers being both experimenters and subjects of study. Sometimes this happens during a preliminary test before a more formal validation of the technology's effectiveness. 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 experiments 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 call such experiments assertions.

However, if the developer is using a new technology on some larger industrial project, we classify it as a case study, since the developer of the technology does not have the same degree of control over experimental conditions.

Field study. A field study may examine data collected from several projects (or subjects) simultaneously. The field study is less intrusive than the case study; otherwise, this method is a form of the replicated experiment we describe later. Since a primary goal is often not to perturb the subject under study, it is often impossible to collect all relevant data in a field study.

Typically, data are collected from each activity in order to determine an activity's effectiveness. Often an outside group will monitor the actions of each subject group. In the case study model, the subjects themselves often perform the data collection activities.

This model best represents an organization that wishes to measure its development practices without changing its processes. An outside group will monitor the subject groups to collect the relevant information. The method also works best for products that are already complete. For example, field study teams can monitor project groups that use a new tool (and ones that do not) in order to determine differences in the effectiveness of what they produce.

Historical methods

A *historical* method collects data from projects that have already been completed using existing data. There are four such methods: literature search, legacy data, lessons learned, and static analysis.

Literature search. The literature search represents the least invasive and most passive form of data collection. It requires the investigator to analyze the results of papers and other documents that are publicly available. This method 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, using a technique called *meta-analysis*.[5]

The literature search method places no demands on a given project and provides information across a broad range of domains. However, a major weakness with a literature search is *selection bias*, which can be characterized as 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 might not be present if the full set of observable data were to be presented.

Quantitative data is often lacking in a literature search because of the proprietary nature of much of this information. Understanding the environment of the published experiment is crucial for interpreting the results.

Legacy data. We often want to understand a previously completed project in order to apply that information to a new project under development. In this method, researchers consider the available data, including all artifacts involved in the product. These artifacts can include the source program, specification, design, and testing documentation, as well as data collected in the program's development stages. There is often 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 (which we describe later).

Study of legacy data can be called a form of *software archaeology*, as researchers examine existing files trying to determine trends. *Data mining* is another term often used for parts of this work as researchers try to determine relationships buried in the collected data. Here researchers are not encumbered by an ongoing project, so costs, schedules, and the need for project delivery are not involved in this activity. All interaction with the project artifacts is passive and is not bound by the real-time pressures of delivering a finished product according to some contractual schedule. Much like a case study, each experiment is unique; it is difficult to compare one project with another because of the great variability of the collected information's availability.

Lessons learned. Researchers often produce lessons learned documents after completing a large industrial project. A study of these documents often reveals qualitative aspects that can be used to improve future developments. If project personnel are still available, it is possible to interview them to understand the effects of methods used.

Such data, however, are severely limited. This form of project may indicate various trends, but cannot be used for statistically validating the results. Unfortunately, lessons learned documents are often "write only"; the same comments about what should have been done are repeated in each successive document. We never seem to learn from our previous mistakes.

Static analysis. In the static analysis method, researchers can often obtain needed information by looking at a completed product. This method is similar to studying legacy data, except that we centralize our concerns on the product developed, whereas studying legacy data includes measuring the development process. In these cases, researchers analyze the structure of the product to determine its characteristics.

Software complexity and dataflow research both rely on this model of analysis. For example, since researchers do not fully understand what the effective measurements are, they assume that products with a lower complexity or simple dataflow will be more effective. They examine the product to learn if its complexity value is lower because of the development method used.

This method is generally a favorite in the academic world, but it is difficult to show that a model's quantitative definition relates directly to the attribute of interest. Program size, for example, is often used as a measure of program complexity, yet numerous studies have shown that the number of lines of code is only marginally related to such complexity.

Controlled methods

A *controlled* method provides for multiple instances of an observation in order to provide for statistical validity of the results. The controlled method is the classical method of experimental design used in other scientific disciplines. There are four types of controlled methods: replicated, synthetic environment, dynamic analysis, and simulation.

Experimentation is one of those terms that is frequently used incorrectly in the computer science community.

Replicated experiment. In a replicated experiment, several projects (or subjects) are staffed to perform a task in multiple ways. Researchers set control variables, such as duration, staff level, and methods used. By using such a method, researchers can establish statistical validity more easily than by relying on case studies.

In a replicated experiment, researchers replace a given task with another task. They might replace Ada with C++, eliminate walkthroughs, or add independent verification and validation. Researchers form several treatments that implement products using either the old or new task. Then they collect data on both approaches and compare the results. This method represents the classical scientific experiment. If there are enough replications—perhaps 20 to 40—researchers can establish the statistical validity of the method under consideration.

The cost of this form of experimentation limits its usefulness. Industrial programmers are expensive; even a small experiment may represent six months to a year of staff time. Because of the enormous cost of replications, these experiments are often limited to a few replications, which greatly increases the risk that the results cannot be duplicated elsewhere.

The effects of performing a replicated experiment among human subjects—such as the development team—disrupt the experiment. Since the various groups know that they are part of a replicated experiment, they may not take their task as seriously as if they were developing a product that would be delivered to a customer. This could have an adverse impact on their care and diligence in performing their tasks, which of course would have an impact on the observed results.

We could avoid this problem by having each replication represent a slightly different product, each one required by a different customer. However, this method then becomes a variation of the case study method described earlier.

Synthetic environment experiments. In software development, projects are usually large and prohibitively expensive. For this reason, most researchers perform software engineering replications in a smaller artificial setting that only approximates the environment of the larger projects. We call these synthetic environment experiments.

Such experiments often appear as a human factors

Table 2. Classification of 612 evaluated papers.

		1985			1990			1995		
		IEEE			*IEEE*			*IEEE*		
Method	**ICSE**	***Software***	**TSE**	**ICSE**	***Software***	**TSE**	**ICSE**	***Software***	**TSE**	**Total**
Not applicable	6	6	3	4	16	2	5	7	1	50
No experimentation	16	11	56	8	8	41	10	3	14	167
Replicated	1	0	0	0	0	1	1	0	3	6
Synthetic	3	1	1	0	1	4	0	0	2	12
Dynamic analysis	0	0	0	0	0	3	0	0	4	7
Simulation	2	0	10	0	0	11	1	1	6	31
Project monitoring	0	0	0	0	1	0	0	0	0	1
Case study	5	2	12	7	6	6	4	6	10	58
Assertion	12	13	54	12	19	42	4	14	22	192
Field study	1	0	1	0	0	1	1	1	2	7
Literature search	1	1	3	1	5	1	0	3	2	17
Legacy data	1	1	2	2	0	2	1	1	1	11
Lessons learned	7	5	4	1	4	8	5	7	8	49
Static analysis	1	0	1	0	0	0	0	0	2	4
Yearly totals	56	40	147	35	60	122	32	43	77	612

study that seeks to investigate some aspect in system design or use. Typically, a large group of individuals—students or programmers, for example—work at some task for several hours. Researchers identify a relatively small objective and fix all variables except for the control method being modified. Researchers often randomize personnel from a homogeneous pool of subjects, fix the duration of the experiment, and monitor as many variables as possible.

A task involving a large group of 20 or 30 people cannot be effectively tested in an experimental setting involving only two or three programmers. The problem of transferring a result covering only a few subjects may not apply to large group studies. Often, researchers conduct such experiments because they are easy to conduct and potentially lead to statistical validity. But we often lose sight of the fact that the experiment itself has little value, since it doesn't relate to problems actually encountered in an industrial setting.

Dynamic analysis. The controlled methods we have so far discussed generally evaluate the development process. We can also look at dynamic analysis methods that analyze the product itself. Many instruments can test a product by adding debugging or testing code so that a product's features can be demonstrated and evaluated when it executes.

For example, a tool that counts the instances of certain features in the source program (such as number of "if" statements) would be performing static analysis. But a tool that executed the program to test its execution time would be performing dynamic analysis.

The major advantage of dynamic analysis is that scripts can be used to compare different products that have similar functionality. The dynamic behavior of a product can be determined often without the need to understand the design of the product itself. Benchmarking suites are examples of such dynamic analysis techniques. Researchers use benchmarking to collect representative execution behavior across a broad set of similar products.

There are two major weaknesses with dynamic analysis. One is the obvious problem that if we instrument the product by adding source statements, we may be perturbing its behavior in unpredictable ways. Also, executing a program shows its behavior for that specific data set, which cannot often be generalized to other data sets. Tailoring performance benchmarks to favor one vendor's product over another's is a classic example of the problems with this method of data collection.

Simulation. Related to dynamic analysis is the simulation method. Researchers can evaluate a technology by executing the product using a model of the real environment. They can then hypothesize how the real environment will react to the new technology. If researchers can model the behavior of the environment for certain variables, they can often ignore other harder-to-obtain variables and glean results more readily using a simulated environment rather than real data.

By ignoring extraneous variables, simulation is often easier, faster, and less expensive to run than the full product in the real environment. Researchers can often test a technology without the risk of failure on an important project. But the real weakness in the simulation method is not knowing how well the synthetic environment models reality. Although we can easily obtain quantitative answers, we are never quite certain how relevant these values are to the problem we are trying to solve.

Which model to use

When we design an experiment, we can collect data that conform to several of our data collection models. The sidebar called "Data Collection Example" shows how we can collect data in evaluating a tool that fits into each of our collection models. In fact, for just about any technology, we can devise a data col-

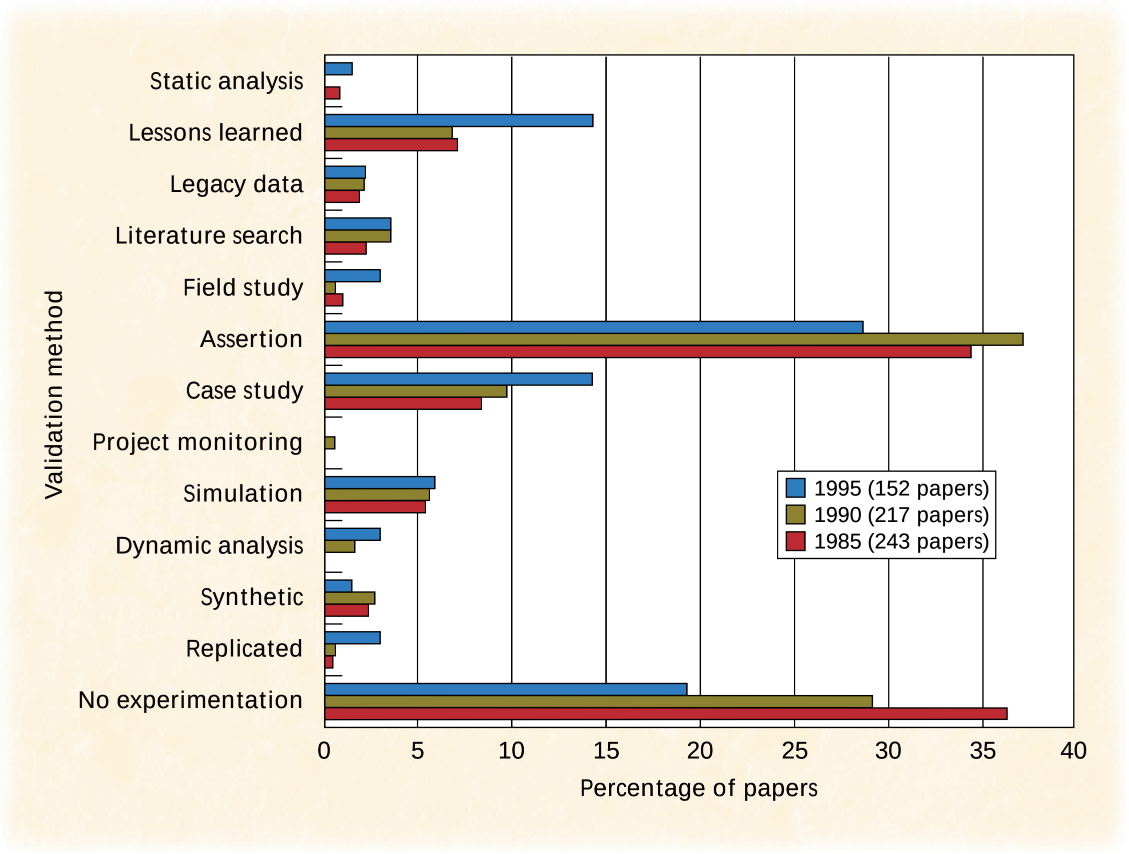


Figure 1. Use of validation methods in 612 published papers.

lection method that conforms to any one of the 12 given data collection methods.

Our 12 methods are not the only ways to classify data collection, although we believe they are the most comprehensive. For example, Victor Basili[6] calls an experiment *in vivo* when it is run at a development location and *in vitro* when it is run in an isolated, controlled setting. According to Basili, a project may involve one team of developers or multiple teams, and an experiment may involve one project or multiple projects. This variability permits eight different experiment classifications. On the other hand, Barbara Kitchenham[7] considers nine classifications of experiments divided into three general categories: a *quantitative experiment* to identify measurable benefits of using a method or tool, a *qualitative experiment* to assess the features provided by a method or tool, and a *benchmarking experiment* to determine performance.

MODEL VALIDATION

To test whether the classification presented here reflects the software engineering community's idea of experimental design and data collection, we examined 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 that 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 the data collection method used to validate the claims in the paper. For completeness we added the following two classifications:

1. *Not applicable.* Some papers did not address some new technology, so the concept of data collection does not apply. For example, a paper summarizing a recent conference or workshop wouldn't be applicable.
2. *No experiment.* Some papers describing a new technology contained no experimental validations.

In our survey, we were interested in the data collection methods employed by the authors of the papers in order to determine our classification scheme's comprehensiveness. We tried to distinguish between data used as a 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 their results.

As in the study by Walter Tichy,[8] we considered a demonstration of technology via example as 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 phase.

The raw data for the complete study—presented in Table 2—involved classifying 612 papers published in 1985, 1990, and 1995.

Quantitative observations

Figure 1 graphically presents the classifications of the 562 papers that we examined. (We assessed 612

Data Collection Example

Several evaluation methods can be used to test a new tool:

- *Project monitoring.* Use the new tool in a project and collect the usual accounting data from the project.
- *Case study*. Use the tool as part of new development. Collect data to determine whether the developed product is easier to produce than similar projects in the past.
- *Assertion*. Use the tool to test a simple 100-line program to show that it can find all errors.
- *Field study.* Distribute the tool across several projects. Collect data on the tool's impact.
- *Literature search.* Find other published studies that analyze the behavior of similar tools.
- *Legacy data.* Find a previously completed project that collected data on using the tool. Analyze this data to see if tool was effective.
- *Lessons learned.* Find a completed project that used this tool and interview participants to see if tool had an impact on the project.
- *Static analysis.* Use a control flow analysis tool to see if one design method results in fewer logic errors than another design method.
- *Replicated experiment.* Develop multiple instances of a module using and not using the tool. Measure the differences.
- *Synthetic.* Have 20 programmers spend two hours trying to debug a module, half of them using the tool and half of them using other techniques.
- *Dynamic analysis.* Execute a program with a new algorithm and compare its performance with the earlier version of the program.
- *Simulation.* Generate a set of data points randomly and then execute the tool alongside another tool to determine effectiveness in finding errors in a given module.

papers and judged 50 to be "not applicable.") The most prevalent validation models appear to be lessons learned and case studies, each at a level of close to 10 percent. Nearly a third of the papers relied on the assertion method. About 5 percent relied on the simulation method, while the remaining techniques were each used in about 1 to 3 percent of the papers. About a third of the papers had no experimental validation; however, the percentages dropped from 36 percent in 1985 to 29 percent in 1990 to only 19 percent in 1995, which seems to indicate improvement.

Tichy 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 percent of papers in other engineering disciplines had no such validation.

Many empirical work papers really are the result of an experiment to test a theoretical hypothesis, so it may not be fair to exclude 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 percentage of design and modeling papers with no validation drops from 50 to about 40 percent in Tichy's study. These numbers are approximate, since we don't have the details of his raw data, but they are consistent with our results.

We have started to investigate how these numbers compare to other disciplines and have looked at various papers in physics, economics, and the behavioral sciences.[9] What we are finding, however, is that papers in archival research journals (such as *Transactions*) do not differ materially from those in other archival journals in the so-called hard sciences.

Qualitative observations

Here are the qualitative observations we made about the 612 papers:

- Authors often fail to state their goals clearly or to point to the value that their method or tool adds to the experimentation process.
- Authors often fail to state how they validate their hypotheses. We had to inspect each paper carefully to determine, as best we could, what the authors were intending to show in the various sections called "validation" or "experimental results." Often such a section couldn't be found, so we had to determine if the presented data could be called a validation.
- Authors often use terms very loosely. Authors would use the term "case study" informally and would even use terms like "controlled experiment" or "lessons learned" indiscriminately. We attempted to classify each paper by what the authors actually did, not by what they called their work. We hope that this article can have some effect on formalizing these terms.

There is one major caveat, however, for understanding the data we present. The papers that appear in a publication are influenced greatly by the publication's editor or, in the case of a conference, by the program committee. In our study, the editors and program committees from 1985, 1990, and 1995 were all different. This difference is a factor that may have affected our outcome. While our goal is to understand how *research* in software engineering is validated, the only way to discover such research is via the *publications* on software engineering, which leads to this dilemma.

In 1992, the US National Research Council panel recommended "…that authors and journal editors attempt to raise the level of quantitative explicitness in the reporting of research findings, by publishing summaries of appropriate quantitative measures on which the research conclusions are based…"[5] Such problems are well known in the software engineering world. Surveys such as Tichy's and our own tend to validate the conclusion that the software engineering community can do a better job in reporting its results.

On the other hand, we need to collect accurate data and avoid British economist Josiah Stamp's lament: "The government is very keen on amassing statistics—they collect them, add them, raise them to the *n*th power, take the cube root and prepare wonderful diagrams. But what you must never forget is that every one of those figures comes in the first instance from the village watchman, who just puts down what he damn pleases."

Through our analysis of some 600 published papers we observed that

- too many papers have no experimental validation at all;
- too many papers use an informal (assertion) form of validation;
- researchers use lessons learned and case studies about 10 percent of the time, with the other techniques being used only a small percent of the time at most; and
- experimentation terminology is sloppy.

While the number of papers with no experimental validation seems to be dropping, clearly more work needs to be done. We want to characterize experimental models according to the types of data they can produce and the types of data industry needs in order to select new technology effectively. Ultimately, we want to enhance researchers' ability to report on software engineering experimentation so that research can better assist industry in selecting new technology. ❖

Acknowledgments

We thank Dale Walters, who helped classify the 600 papers used to validate the models described in this article.

References

1. W.R. Adrion, "Research Methodology in Software Engineering: Summary of the Dagstuhl Workshop on Future Directions in Software Engineering," *SIGSoft Software Eng. Notes*, Vol. 18, No. 1, ACM Press, New York, 1993, pp. 36-37.
2. S.L. Pfleeger, "Experimental Design and Analysis in Software Engineering," *Annals of Software Eng. 1*, Baltzer Science Publishers, The Netherlands, 1995, pp. 219-253.
3. M.V. Zelkowitz et al., "Software Engineering Practices in the United States and Japan," *Computer*, June 1984, pp. 57-66.
4. V.R. Basili and H.D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Trans. Software Eng.*, 1988, pp. 758-773.
5. *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 D.C., 1992.
6. V.R. Basili, "The Role of Experimentation: Past, Present, Future," (keynote presentation), *Proc. Int'l Conf. Software Eng.*, IEEE CS Press, Los Alamitos, Calif., 1996.
7. B.A. Kitchenham, "Evaluating Software Engineering Methods and Tools," *SIGSoft Software Eng. Notes*, ACM Press, New York, 1996, pp. 11-15.
8. W.F. Tichy et al., "Experimental Evaluation in Computer Science: A Quantitative Study," *J. Systems and Software*, 1995, pp. 9-18.
9. M.V. Zelkowitz and D. Wallace, "Experimental Validation in Software Engineering," *Information and Software Technology*, Vol. 39, 1997, pp. 735-743.

Marvin V. Zelkowitz is a professor of computer science at the University of Maryland, codirector of the Fraunhofer Center Maryland, and, until recently, a faculty researcher at the National Institute of Standards and Technology. He is also one of the directors of the NASA Goddard Space Flight Center Software Engineering Laboratory, which has been studying the effects of software engineering technologies in practice since 1976. His research interests include environment design, empirical studies, and measurement. Zelkowitz received an MS and a PhD in computer science from Cornell University. Contact him at mvz@cs.umd.edu.

Dolores R. Wallace is a computer scientist in the software quality group at the National Institute of Standards and Technology and leads the Reference Data: Software Error, Fault, Failure, Data Collection, and Analysis Repository Project, which provides metrology and reference data for software assurance. Her research interests include methods and tools, especially for verification and validation, to improve software quality and correctness. Wallace received an MS in mathematics from Case Western Reserve University. Contact her at dwallace@nist.gov.