EXPERIMENTAL MODELS FOR SOFTWARE DIAGNOSIS

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

Dolores Wallace
NIST/CSL
Gaithersburg, MD 20899

NISTIR 5889

Abstract

Experimentation and data collection are becoming accepted practices within the software engineering community to determine the effectiveness of various software development practices. However, there is wide disagreement as to exactly what the term "experimentation" means in this domain. It is important that we understand this concept and identify how we can best collect data needed to validate software methods that are effective. This understanding will provide a basis for improved technical exchange of information between scientists and engineers within the software community.

Keywords: Data collection; Experimentation; Measurement; Process measures; Product measures; Software complexity;

Contents

1 Experimentation

Experimentation[1] and data collection are becoming accepted practices within the software engineering community as a means to understand both software and the methods used in its construction. Data collection is central to the NASA/GSFC Software Engineering Laboratory [5], 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) [17]. However, there are many ways to collect information within the software engineering community. The purpose of this paper is to explore these methods and to understand when each is applicable toward our understanding of the underlying software development process. The long term objective is to enable a better technical exchange of information between scientists and engineers in the software community. The software industry will have the means to become aware of software technology and criteria for selecting the methods and tools appropriate to their development projects.

In this section we explore the goals for experimentation and describe the context of why we want to collect data, and in Section 2 we describe the various forms of experiments that can be applied to software development, what data we can collect by this experimentation, and the strengths and weaknesses of the various methods. In Section 3 we compare our classification model to other models of software engineering experimentation, and in Section 4 we apply this model to a collection of some 600 published papers in order to understand how experimentation is being used by the software engineering community. We then compare our results with a related study performed by Tichy [20], who also looked at the role of experimental validation in published papers.

1.1 Goals for Experimentation

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. Just saying 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 reliability, we have measures that are not as clear as in the previous example. We can use measures like failures in using the product per day, errors found during development, or MTBF (Mean Time Between Failure) as used in hardware domains. All of these give some indication of the relative reliability between two products, but the correlation between these measures and the attribute we want to evaluate is not as clear cut. For example, a count of the number of errors found during testing does not, by itself, indicate if there are further errors remaining to be found. Unfortunately,

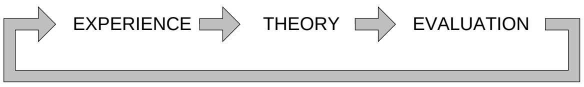
[1] Contribution of the National Institute of Standards and Technology. Not subject to copyright.

for other applications of effectiveness, we have no good measurement scale. Safety is one example of this measure. What does it mean for one product to be safer than another? Safety is related to reliability, but it is not the same. A very unreliable program can be very safe if it can turn itself off each time the software fails. Security is another one of these nebulous attributes. What does it mean for one product to be more secure than another? Does it mean how long 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?

While the classification of attributes and their effective measurement is an important aspect of software engineering, we do not address this problem further in this paper. We will simply assume that we have some mechanism for measuring the attribute we wish to observe in our experiment.

We summarize this then by saying that within the software engineering community, 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. Questions that the experimentalists need to address include: Does a particular technique work? Is the resulting software more effective? What processes can use these techniques? What is the quantitative improvement in our "effectiveness" by using this technique (i.e., can we measure the improvement in the measurement of interest)?

Aside from validating the effectiveness of particular techniques, experimentation provides an important feedback loop for the *science* of software engineering, given by (c.f. [19]):



Experimentation is a crucial part of the evaluation feedback loop in the development of new theories. Can we modify the underlying theory upon which the technique is based? What predictions can we make upon future developments based upon using these techniques?

1.2 Measurements

Software development needs to generate reliable, effective products. For most engineering disciplines we have various performance measures that can be used in this evaluation. For example, if a highway is built, there are tests that can be made on the concrete or asphalt road to test durability and other surface characteristics; for bridge building we can test for the load the bridge can handle and for the various stresses that the bridge may be subject to. However, for a software product, we have few related performance measures. We have no qualitative measure of a program that is generally agreed to by most professionals.

We cannot measure the reliability (in absolute terms) of the software or the correctness of the software. Because of this, given data we can measure, what are the appropriate conclusions we can draw from this data? Since we don't have causal relationships between data we can measure and the attributes we are interested in, we often have to resort to indirect measures. Rather than measuring the product itself, we often measure the process used to develop the product. We make the assumption (which has never been thoroughly evaluated) that an effective development process will produce a good product. This assumption is the basis for the Capability Maturity Model and the interest in ISO 9000 certification, and while they intuitively seem reasonable, still need scientific validation.

Because of the lack of effective measurements on a software program, we generally approach the problem of experimentation with software development in two diverse ways: (1) We can measure software products and develop a quantitative evaluation of the components that go into that piece of software; or (2) We can develop a quantitative evaluation of the process used to develop the product. Both serve as methods for experimenting on software development to result in a better understanding of the relationship between development and the attribute we are trying to quantitatively measure.

1.3 Process and Product

As we just described, experimentation in software engineering can be separated into experiments validating a particular product and experiments validating a particular process. Depending on the specific underlying theory, measurement of program complexity is defined on the basis of measures such as lines of code, function points, or cyclomatic complexity. In these examples, the artifact of study is the program itself, or a representation of the program (e.g., a specifications or requirements document).

On the other hand, measures like reliability (e.g., number of failures in the software or mean time between failure), cost of development, and time to develop, all represent measures of the development process. They do not represent data about the artifact itself, but instead represent information collected by the group performing the development or use of the product.

Can we modify the development process (e.g., replace testing with a verification process, or replace Ada with Smalltalk as the development language) and then measure the effect of this change on the overall development process with respect to measures like cost and reliability? If our measures were effective, then the changes in our measurement data would indicate changes in actual development methods.

However, what we would really like to know is what effect do changes in the development method (e.g., changing methods to test the product) have on the product itself? If changing a programming language, for example, results in more errors being found during the testing phase (generally perceived to be a good thing), does this mean that the new language results in more errors being caught during development (and hence results in a more reliable product), or does it mean that the new language results in more errors being made by the development staff (and hence there may be many more latent errors not yet found, resulting in a less reliable product)?

For the most part, we haven't established this connection between the development process and the product being developed, but we assume they behave as highly correlated phenomena. Good testing strategies, for example, are viewed as producing well tested products containing few errors.

We have been fairly informal, so far, in using terms like process, techniques, and methods. These terms are quite overworked and mean different things to almost everyone. For the remainder of this paper we will use ISO 12207 [10] terminology wherever possible. This terminology is:

- The major work items for software engineering are *processes*. There is an acquisition process, a development process, an operations process, a quality assurance process, etc.
- Each process consists of a set of *activities*, or a set of jobs to be done. For software development, the activities consist of the typical waterfall lifecycle activities, such as software design, coding, and testing activities. For the most part, experimentation is too specific to affect an activity. We generally view the set of activities as fixed and are most concerned about how to achieve each activity most effectively.
- Activities are broken down into *tasks*. These now vary depending upon the particular method employed. For example, the coding activity may consist of the source code generation task, the unit testing task, and the documentation task. For a cleanroom development [14], the unit testing task may be replaced by a verification task. An inspection task or a walkthrough task may be added. It is at this level that experimentation may result in data being collected which can be used to compare one task with another.
- Tasks can be further subdivided into *methods*, *techniques*, and *tools*. For this report we will not differentiate between a method or a technique. Both represent a relatively specific algorithm for achieving the goals of the task. For example, path testing, branch coverage, and mutation testing are all methods for achieving a unit testing task. They have different effectiveness, and a considerable number of experimentation papers have been published about the relative merits of each method toward realizing an effective testing task.
- We will describe a *tool* as a program which implements a method (or technique). Thus a program which takes a module, creates mutations of it, and then executes each mutation, is a tool which implements the mutation testing method. A long range goal for much of software engineering is to implement every effective method as a tool. This would go a long way toward automating the software development process.

We therefore view experimentation as dependent upon collecting data that can be used to differentiate among various tasks (and the components of tasks) in order to determine which tasks produce more effective programs than other tasks.

1.4 Why study process?

At first glance it may seem superfluous to study the process of software development. In most engineering domains we are concerned about the product being designed. For example, new aircraft design seemingly depends upon airframe composition, engine design, and on-board avionics for control, and bridge design depends upon the composition of the components used and the basic model of the bridge (e.g., suspension, cantilever, truss, arch). The process used to construct the product does not seem to be involved. However, a more in-depth study of other engineering disciplines reveals many similarities with the software engineering domain, the major difference being that most other engineering domains are older and have undergone this process evolution research many years earlier.

Leveson [12] provides an excellent example comparing the development of steam boilers in the 19^{th} century and the development of software engineering today. The basic problem was simple: As the temperature and pressure increased in order to increase the power and efficiency of the steam that was produced, boilers had a tendency to explode and kill nearby people. While engineers were concerned with the development of effective *products* that did not explode, the *process* of building and using boilers was also under considerable study. Leveson cites the following process issues that occurred during the development of boiler technology:

1. As a way to limit explosions, there were proposals to limit boilers to low pressure, and hence, low power production. This was an attempt to modify the process where the boiler would be used. The analogy today would be to determine the application domains where a given software technology could be used effectively.

2. Since boilers were being designed by poorly trained engineers and used by unskilled operators, there were attempts to limit where they could be used. The obvious economic benefit (i.e., measurement of power production) made this process solution ineffective. Cleanroom is a modern-day example of this phenomenon. Although hard to implement, data on its use has shown it to be too effective to be ignored [3].

3. Liability laws were changed to allow for compensation to families of passengers killed in boiler accidents. The economic measurement of potential loss was a strong impetus to the process of construction of boilers to include safety features that would limit losses. Unfortunately, perhaps, the software industry has not reached this level of concern. In fact, the opposite is often the case. Companies are often unwilling to investigate certain technologies (e.g., security and safety issues) since they are not liable as long as "due diligence" was used in the construction of the software. If better technologies were used and they knew about these problems, they would be liable for not producing better products.

Leveson cites several reasons that drove the problems with boiler technology in the 1800s. "Safety features [of boilers] were not based upon scientific understanding of the causes of accidents." But "trial and error is a time-tested way of accumulating engineering knowledge," and experimentation did lead to more effective design of products.

1.5 Experimentation – Not!

Experimentation is one of those terms frequently used incorrectly in the computer science community. Papers are written that explain some new technology and then "experiments" are performed to show the technology is effective. In almost all of these cases, this means that the creator of the technology has implemented the technology and shown that it seems to work. Here, "experiment" really means an example that the technology exists or an existence proof that the technique can be employed. Very rarely does it involve any collection of data to show that the technology adheres to some underlying model or theory of software development, or that it is effective, as "effective" is defined previously, to show that application of that technology leads to a measurable improvement in some relevant attribute.

A typical example could be the design of a new programming language where the "experiment" would be the development of a compiler for the new language and sample programs compiled on this compiler. The "success" for the experiment would be the demonstration that the compiler successfully compiles the sample programs. What is missing is data that shows the value or effectiveness of this new language.

A true experiment would determine whether programs written in this new language were more effective than programs written in another language, executed faster, were easier to develop, easier to maintain, or fulfilled some other attribute of interest to the language designer. Simply building and demonstrating a compiler does not really address the utility of that language within the realm of software development.

Without the true experiment, why should industry select a new language (or new method or tool, etc.)? On what basis should another researcher enhance the language (or extend a method) and develop supporting tools? As a scientific discipline we need to do more than simply say, "I tried it and I like it" [7]. Can we imagine an engineer at an airplane manufacturer going to the head of the company saying "Yesterday, we discovered this new metal. Beginning tomorrow all airplanes will be built using it, and we don't need to test it. It obviously works." However, such statements are said daily in almost every information technology company, and the resulting engineer not only does not get fired, but probably gets a bonus instead.

1.6 So, How Do We Experiment?

This then leads to the purpose of this paper. How do we collect data necessary to evaluate the effectiveness of the tasks that are part of a software development activity? How do we validate a new development technique? Can we use this data to determine if a new product is effective? What are the characteristics of a new product? Is it reliable? Is it complex? Is it efficient? Understanding the answers will lead to the goal of identifying the elements needed in experiment design and data collection relevant to types of hypotheses, which in turn will yield uniform methods of representing results that will enable industry to understand and compare techniques and tools.

When one thinks of an "experiment," one often thinks of a roomful of subjects, each being asked to perform some task, followed by the collection of data from each subject for later analysis. In a recent report [16], experimentation is broken down into three categories: case studies, academic studies, and industrial studies. However this narrow definition does not distinguish among the various forms of data that may be collected. The selection of statistical methods for measuring and expressing the results of software experiments depends on the various forms of data that may be collected.

These categories, however, are really related methods among the four models of experimentation [1]. Experimentation can mean any of the following approaches:

1. *Scientific method.* This, as described above, is the "classical" way to run an experiment. A theory to explain a phenomenon is developed. A given hypothesis is proposed and then alternative variations of the hypothesis are tested and data collected to verify or refute the claims of the hypothesis.

2. *Engineering method.* In this model, a given solution to a hypothesis is developed and tested. Based upon the results of the test, the solution is improved, until no further improvement is required.

3. *Empirical method.* In this model, 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.

4. *Analytical method.* A formal theory is developed, and results derived from that theory can be compared with empirical observations.

Note that not all of these require the collection of data coincident with the running of the experiment. While this is typically true of the scientific method, others, such as the engineering method and empirical method, can use previously-collected historical data for appropriate validation.

The common thread from all of these questions is the collection of data on either the development process or the product itself. We will address this in the next section by describing several classes of data collection models that can be designed to test hypotheses like those suggested previously. From the data collection models, we will describe classes of experiments that can employ these data collection methods.

2 Data Collection Models

In this section we discuss the various data collection models that are used in the software engineering domain and will discuss how they relate to the various experimental models explained in the previous section. Importance, time, budget, and criticality to project success all determine which one to employ for a given development.

2.1 Experimental design

When we do an experiment, more properly an experiment using the scientific method described above, 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* 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 we are concerned about compared to some other treatment.

In developing an experiment to collect data on this attribute, we have to be concerned with three aspects of data collection [18]:

1. *Replication* – The most important attribute of the scientific method is to be able to replicate the results of an experiment so that other researchers can reproduce the findings of a given experiment. In order to ensure that this is so, we must be certain that we don't *confound* two effects. That is, we must make sure that unanticipated variables are not affecting our results.

   As a simple example, if our subjects are student programmers who use a certain method and professional programmers who use another, we cannot be sure whether method or experience is the cause of any differing results. In this case we are confounding the impact of experience and method in achieving our results.

   We counteract this confounding effect by *randomizing* the factors that we are not concerned about. If we mix up students and professional subjects for each method, and we still observe a difference, then we can say (with a fair degree of certainty) that the method and not the experience was the cause of the difference.

2. *Local control* – Local control refers to the degree to which we can modify the treatment applied to each subject. This is one of the major distinguishing characteristics separating a case study from a formal experiment. We usually have little control over the treatment in a case study.

   In a *blocking* experiment, we assume each subject of a treatment group comes from a homogeneous population. Thus if we randomly select subjects from a population of students, we say that we have a blocked experiment of students. For ease of mathematical analysis, we often try to *balance* each treatment by having an equal number of instances for each assignment to the factors.

3. In a *factorial design* we apply for each factor every possible treatment. Thus if there are two factors we wish to evaluate, and each has 3 possible values, then we need to run 9 experiments, with subjects randomly chosen from among the blocked factors.

   In a factorial design, however, we often can test for several factors at once. If we have two factors, A and B with two values each, then half the treatments can be viewed as factor A with one value and half with factor B with the other value. Similarly, we can do the same with factor B. With 32 subjects, we can test for four separate factors, each with two values, and each replicated twice. Although each individual treatment has only two subjects, each factor is replicated 16 times in this design.

In addition to the above three traditional characteristics for an experiment, we add the following two that seem appropriate for the software development domain.

Influence. We want to know the impact that a given experimental design has on the results of that experiment. We will call the various methods *passive* or *active*. Passive methods are those that view the artifacts of study as inorganic objects that can be observed, inspected, and studied with no effects on the object itself. Static analysis techniques which look at the set of collected data artifacts are examples of a passive method.

On the other hand, we will consider active methods as 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. For the most part these are time-sensitive studies where we are collecting data in real-time to monitor either the process being employed or the product that is being developed. The well-known "Hawthorne effect," where the process of observing individuals causes them to change their behavior, is an example of an active effect we wish to minimize.

Results obtained via passive methods should be more indicative of the actual development process due to a lack of interacting with the developers. However, passive methods, by their nature, rarely collect exactly the data needed to arrive at the desired conclusion. Therefore, we usually need to employ active methods, with an attempt to eliminate, minimize, or at least understand, the influence the data collection process has on the project.

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.

We will classify the various data collection models and indicate how they are affected by the various experimental parameters. We will group the various data collection methods into three broad categories:

1. An *observational* method will collect relevant data as a project develops. In general, there is relatively little control over the development process other than using the new technology that is being studied.
2. An *historical* method collects data from projects that have already been completed. The data already exists; it is only necessary to analyze what is already there.
3. 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.

2.2 Observational Methods

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

2.2.1 Project Monitoring

Project monitoring represents the lowest level of experimentation and measurement. It is simply the collection and storage of data that occurs during project development. We view it as a passive model since we are simply collecting whatever data the project is generating with no attempt to influence or redirect their process or affect the data they are collecting.

Method. Almost every project collects some data, often in the form of time cards recorded by project personnel on a weekly basis or a list of error report forms submitted by users. This project data should be collected by the organization and made available to others for later use.

Surprising as it seems, many organizations do not preserve project information. In a survey performed in the early 1980s [21], it was found that although project information was often collected by project management, this information was also "owned" by the project manager and would not be used on some future project. The situation is still quite true today in many organizations.

The argument that data collection and preservation uses valuable computer resources simply is no longer true. With 1.6 Gbyte hard disks for PCs costing under $300 today, there is no reason not to collect this information and preserve it for future use.

Strength. Project monitoring represents the minimal level of data collection that should be applied to any organization. The data is generally collected, anyway. The only problem is the centralization of the collection process and the capability to retrieve this information later. This solution requires some minimal coordination among the various development activities in an organization, but represents a level of experimentation that should be required of all.

Weakness. This method generally lacks experimental goals or consistency in the data that is collected. Data is simply collected for its own sake. 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 are crucial for later process improvement activities, such as applying Basili's Quality Improvement Paradigm (QIP) [4]. One first has to know where one is, before trying to determine the effect of changes.

Since there is no design of what data to collect, consistency between projects is often limited. Conclusions based upon data from several such projects must be suspect since it is not clear how each project interpreted its data.

2.2.2 Case Study

In a case study, a project is monitored and data collected over time. The project is often a large commercial development and would be undertaken whether data was to be collected or not, although could also be part of a university project as well. The key factor is that the development would happen regardless of the desire to collect data for this study. With a relatively minimal addition to the costs to the project, valuable information can be obtained on the various attributes characterizing its development.

This method differs from the project monitoring method above in that data collection is derived from a specific goal for the project. A certain attribute is monitored (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 more consistent baseline is built that represents the organization's standard process for software development. Once the baseline is established, it is easier to measure the effects of any change caused by experimentation with a new technology.

While project monitoring is considered passive, a case study is an active method because our experimental goals influence the data we want to collect, which may have an effect on the development process itself. The very nature of filling out a certain form, which, by itself, may not be very intrusive to the development group, may have the side effect of having the staff think about certain issues in order to fill out the form. This could cause them to react differently had they not filled out that form.

One of the goals of groups like the NASA Software Engineering Laboratory is to establish a consistent project monitoring data collection activity to reduce this case study method to a passive project monitoring method. This accomplishment would move the results obtained from an active method to a passive method where the results may be more significant, since we are no longer perturbing the process in our attempts to collect the relevant information.

Method. Typically, data collection forms are used to perform case studies. Resource data (e.g., hours worked) is collected from project personnel and often forms are collected periodically (e.g., submission of forms to identify errors, when modules are placed under configuration control, when new releases are made). This data can be processed in a database to produce a profile of information that describes the behavior of the development over time.

Strength. 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. With only minimal changes to accounting data already collected by many companies, valuable development data can be collected to produce these profiles.

There are many developments currently happening, so if the organization is attuned to the needs for experimentation and data collection, data from many projects can be amassed over a short period of time.

Weakness. 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 becomes difficult. There have been some efforts at collecting different profiles and looking at techniques such as cluster analysis [13] or optimized set reduction statistical techniques to combine diverse projects. However, the lack of experimental controls to set common sets of independent and dependent variables for each such case study limits their utility.

In addition, most projects may involve a significant expenditure of money. The NASA SEL has estimated that collecting data for such projects may require from 5% to 6% additional funds. While a relatively low percent of total projects funds, this amount may represent, in absolute numbers, a large amount of money on a large project. When a project is late or over budget, the data collection activity is often the first one curtailed. And more importantly, this late project is exactly the project where we must have data for analyzing what went wrong to avoid similar problems in the future.

Because case studies are often large commercial developments, experimental controls are often hard to impose. 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, with appropriate reliability, may often mean that experimental goals must be sacrificed. Experimenting with new development methods or new tools may be a risk, which management is not willing to take, so good experimental results become hard to achieve.

If data is collected for later study, there will be further incentives to curtail such activities if behind schedule or over budget. Those projects which merge experimentation for research with real-time feedback of data to project management stand the best chance of collecting data that will be useful with data collection activities continued in the face of adverse conditions.

2.2.3 Assertion

We subdivided the case study method by adding an assertion classification. This is a case study where the developer of the technology is both the experimenter and often the subject of the study. The purpose of this case study is to validate the effectiveness of the proposed technology (e.g., building a toy program to show how well a new tool works). In essence, the developer is saying "I tried it, and I like it." However, if the developer is using a new technology on some larger industrial project, we will classify it as a case study since the developer of the technology does not have the same degree of control over the experimental conditions that need to be imposed.

Method. The developer of a technology uses the new technology on a project. In general, the sample project used is relatively small or incomplete. In some cases, the purpose of the development is to simply show the value of the new technology and has no intrinsic value itself.

Strength. The developer of a technology knows it best and is often best able to demonstrate its effectiveness.

Weakness. While the assertion form of case study has value, it is often viewed with more suspicion in other scientific disciplines than is a case study by some other neutral party. The potential for biasing the results by using a specific example is great.

2.2.4 Survey

It is often desirable to compare several projects simultaneously. This is related to the case study, but is less intrusive to the development process. For this reason, we classify it as a passive method and not as an active method. If the survey becomes very intrusive with a significant involvement of the development staff in the collection of the necessary data, then this method is just a form of the replicated experiment to be described later.

Method. This form of experiment represents a parallel set of case studies. Typically, survey forms are collected from each activity in order to determine the effectiveness of that activity. Often an outside group will come and monitor the actions of each subject group, whereas in the case study model, the subjects themselves perform the data collection activities.

Strength. This model best represents an organization that wishes to measure its development practices without changing the process to incorporate measurement. An outside group (either another organization or another group within the same company) will come and monitor the subject groups to collect the relevant information.

The method also works best for products that are already complete. If a new tool has been established in one organization, survey teams can monitor groups that use the new tool and ones that do not in order to determine differences in the effectiveness of what they produce.

Weakness. The model limits the information that can be collected. Since a primary goal is often not to perturb the activity under study, it is often impossible to collect all relevant data.

2.3 Historical Methods

An *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.

2.3.1 Literature Search

The literature search represents the least invasive and most passive form of data collection research. It requires the investigator to read and analyze the results of papers and other documents that are generally available to the general public. 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.

Method. This method is included for completeness of the set of given methods. Relevant papers in journals and conference proceedings are read, and the results are synthesized into an approach that may be used in another environment. This is the classical form of information gathering using a library as the main source of information.

Strength. This method places no demands on a given project and only affects the person doing the literature search. It provides information across a broad range of industries, and access to such information is at a low cost. In addition, since the uncertainty of a measurement drops with an increasing number of observations, combining the data from several previously published projects into one larger analysis (e.g., *meta-analysis*) provides for a more precise finding [15].

Weakness. 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.

Quantitative data is often lacking due to the proprietary nature of much of this information. Often the controls used in the experiment are lacking leading to an inability to duplicate the experiment at another location. This makes it difficult to determine homogeneity among several reported data items. The amount of information is limited and its validity is often open to question. Understanding the environment of the published experiment is crucial for interpreting the results, and such an understanding is often lacking.

2.3.2 Study of Legacy Data

We often want to determine what happened on a previously completed project, so that we can apply this information and evaluate a new project now under development. The study of legacy code and the data collected with its development is often used to validate experimental results on this new project. This is similar to an after-the-fact project monitoring method. Rather than collecting data as a project proceeds, the data is collected from projects that have already been completed.

Method. Literally billions of lines of code exist worldwide, and much documentation and data exist for many of these billions. A search through this documentation often discovers interesting aspects that may be applicable to new development methods.

In this method we consider the available data to include all artifacts involved in the product. These artifacts can include the source program, specification, design, and testing documentation, as well as data collected in its development. We assume there is a fair amount of quantitative data available for analysis. Where 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.

Strength. Study of legacy data is a low cost form of "experimentation." The term experimentation is used loosely here. In this case, existing documentation and data are studied to see if any effects can be traced to the methods used in its development. 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.

In this case, we are not encumbered by an ongoing project, so costs, schedules, and the needs for delivery of a product are not involved in this activity. All interactions with the project artifacts are passive and are not bound by the real-time pressures of delivering a finished product according to some contractual schedule.

Weakness. Since data has already been collected, the information available is necessarily fixed and limited. It is not possible to collect information not thought of originally. This limits the results we can obtain from these types of experiments. Much like a case study, each experiment will be unique and it will be difficult to compare one project with another due to great variability in the availability of the collected information.

2.3.3 Study of Lessons-learned

A weaker form of legacy data is the investigation of lessons-learned documents from previous projects. If project personnel are still available, it is possible to obtain some low-cost trends in looking at the effects of methods. This method is more of a qualitative assessment of a completed project.

Method. 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 aspects which can be used to improve future developments.

Strength. This study is perhaps the cheapest form of experimentation. Only existing documents are studied, and interviews with project personnel may indicate other existing conditions. It is most effective for recently completed projects where the project personnel are available for interviews and memories are all fresh.

Weakness. By its nature, the data available is severely limited. This form of project may indicate various trends, but cannot be used for statistical validity of the results. Unfortunately, lessons-learned documents are often "write only," and the same comments about what should have been done is repeated in each successive document. We never seem to learn from our previous mistakes.

2.3.4 Static Analysis

The passive methods we have described so far all evaluate development according to some criteria. We can often obtain needed information by looking at the completed product, which we call the static analysis method. This is a special case of studying legacy data except that we centralize our concerns on the product that was developed, whereas legacy data also included process measurement. In these cases, we analyze the structure of the product itself to determine characteristics about use. The realms of 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.

The use of tools to analyze software points to a possible ambiguity in our classification model. Tools can be used to help develop software (e.g., use a data flow tool to find illegal usage of a variable) or to help evaluate software (e.g., use a cyclomatic complexity tool to classify the complexity of each module in a system). In some cases, the same tool can be used for both purposes. In this paper, however, we are only concerned about the latter application of collecting data about a completed project. Using tools during the construction of software is an important, but for this paper, unrelated issue.

Method. A product, usually as a collection of source program files, is analyzed by a series of tools in order to extract relationships among the components of the program. Measures (e.g., the software science measures, cyclomatic complexity, lines of code, function points, prime decompositions, fan-out, data-bindings) are computed in order to determine a value for the program according to that measure. The assumption is made that programs with a better value will be easier to understand and process. Therefore, products that have better computed attributes should be more reliable, more maintainable, etc. and be more effective products according to our usual set of attributes. Also methods which produce such products will be more effective methods.

This method, because of its ease to implement, is generally a favorite in the academic world in demonstrating that a given measure has a positive correlation to an attribute of interest. Most correlations, however, are relatively weak.

Strength. Most program analysis measures are variations of software complexity models that are based upon specific models or formulas, so the computation of these complexity values is often specific and reproducible. Since the product is complete, it has the same positive attribute of studying legacy data in that the analysis is not influenced by the needs of an ongoing project.

Weakness. It is difficult to show that the model's quantitative definition relates directly to program complexity. Numerous studies have shown, for example, that lines of code is only marginally related to program complexity. The application domain also has a big impact. All the other measures cited above also have problems in addressing the attribute they claim to address.

2.4 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.

2.4.1 Replicated Experiment

The replicated experiment represents the other extreme of active method from the case study. Several projects are staffed to duplicate a given specification in several ways. 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 generally know they are part of an experiment and not part of a true development.

Method. In a replicated experiment, a given task is proposed to be replaced by another task (e.g., replace Ada by C++, eliminate walkthroughs, provide for independent verification and validation as part of acceptance testing). Several groups are formed to implement products using either the old or new task. Data is collected on both approaches, and the results are compared.

Strength. This represents a true scientific experiment in a realistic setting. If there are enough replications, statistical validity of the method under study may be established. Since this is part of a realistic setting, the transfer of this technology to industry should be apparent, and the risk of using the results of this study should be lessened.

Weakness. The cost of this form of experiment limits its usefulness. Industrial programmers are expensive and even a small experiment may represent 6 months to a year of staff time. Since we need about 20–40 replications to ensure good statistical validity of our results, the total costs for such an experiment can be enormous. In this case, replications are often limited to at most 2–4, which greatly increases the variability of the results and the lack of statistical significance in its conclusions.

In addition, the effects of performing a replicated experiment among human subjects (i.e., the development team) perturb 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.

Of course, we could avoid this by having each replication represent a slightly different product, each one required by a different customer. This then becomes a variation of the case study method described earlier. That method has its own set of strengths and weaknesses, as we previously described.

2.4.2 Synthetic Environment Experiments

Another form of widely used experiment is the synthetic environment experiment. It is similar to smaller versions of the replicated experiment above. In the software engineering community, this often appears as a human factors experiment investigating some aspect in system design or use. Typically, a large group of individuals (e.g., students or sometimes industrial programmers) work at some task for several hours, leading to data being collected on this task.

Like the replicated experiment, it can be used to obtain a high degree of statistical validity, but even more than the replicated experiment, may perturb the results to make their applicability to an industrial setting very suspect.

Method. 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.

Strength. The large number of subjects involved in such an experiment greatly leads to statistical validity of the results. Such experiments are often modeled after psychological experiments, where there has been a great body of research on how to conduct such experiments and how to analyze the data collected from these activities.

Weakness. By its very nature, such experiments are often of short duration. Since we are dealing with many subjects, the time and cost involved in each subject is relatively low. So while we can get a high degree of statistical significance by studying the behavior of even hundreds of subjects, the problems they are working on will be of limited usefulness, relevance, or importance to some of the complex problems encountered in building large systems.

Because the objectives of such experiments are often limited, the relevance of transferring the results of such experiments to industry may also 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 realistic.

In addition, tasks that by their nature are complex may not be tested in this manner. Developing good configuration management, for example, requires a large system, so an experiment of a few hours may prove little. An experiment requiring an analysis of a system of 100 modules over 4 hours may not translate to the management of a system of 10,000 modules. Similarly, a task involving a large group of 20 or 30 people cannot be effectively tested in an experimental setting involving 2 or 3. The scaling-up problem of transferring a result covering a few subjects may not apply to large groups of individuals.

The scaling-up issue points out a major weakness in this model of experimentation. Often such experiments are conducted because they are easy to conduct and should lead to statistical validity. We often lose sight of the fact that the experiment itself has little value since it doesn't relate to any problems actually encountered in an industrial setting.

2.4.3 Dynamic Analysis

The controlled methods we have so far discussed all evaluate the development process. We can also look at controlled methods that execute the product itself. We call these dynamic analysis methods. Many instrument the given product by adding debugging or testing code in such a way that features of the product can be demonstrated and evaluated when the product is executed. Others execute the product as a means to compare it with other products. This differs from the static analysis method mentioned earlier; in that instance the product is only evaluated as is.

For example, a tool which counts the instances of certain features in the source program (e.g., number of **if** statements) would be a static analysis of the program, whereas a tool which executed the program to test its execution time would be a dynamic analysis method. We discuss this further in Section 2.5.

Method. 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.

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

Weakness. 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. Secondly, as Dijkstra had observed close to 30 years ago, testing a program shows the presence of errors and not their absence [8]. Similarly, in this case, executing a program shows its behavior for the specific data set being used that cannot often be generalized to other data sets. The tailoring of performance benchmarks to favor one vendor's product over another is a classic example of the problems with this method of data collection.

2.4.4 Simulation

Related to dynamic analysis is the concept of *simulation*. In this case we evaluate a technology by executing the product using a model of the real environment. In this case we hypothesize, or predict, how the real environment will react to the new technology.

Method. This process is much like the dynamic analysis method given above with the difference being that we execute the product using a simulated environment rather than real data. 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.

Strength. By ignoring extraneous variables, a simulation is often easier, faster, and less expensive to run than the full product in the real environment. We can often test a technology without the risk of failure on an important project, and we will not be adversely affected by the needs of project personnel to complete a project.

Weakness. The real weakness in a simulation is a lack of knowledge of how well the synthetic environment we have created 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.

2.5 Which model to use

When we wish to collect data from an experiment, it is not sufficient to consider only the object that is under study. For example, if the goal is to test the impact of a new testing tool, one cannot a priori decide what sort of data must be collected. Data can be collected that conforms to several of our data collection models. All of the following represent different ways to model data collection activities related to this new tool:

- *Case study.* Use the tool as part of new development and collect data to determine if the use of the tool results in a product that seems more efficient, more reliable, or easier to develop than similar projects were in the past.
- *Assertion.* Use the tool to test a simple 100 line program to show how easy it is to find all errors.

- *Literature search.* Determine if there are any other published studies that analyze the behavior of tools such as the testing tool under evaluation.
- *Legacy data.* Find a previously-completed project that collected data on using the tool to determine whether the characteristics of the project improved because of the use of the tool.
- *Lessons learned.* Find a completed project that used this tool and ask the participants of that project if they believed the tool had a positive impact on the project and why.
- *Replicated experiment.* Develop multiple instances of a module in an industrial setting, some using the tool and some not using it, to see if there are any measurable differences.
- *Survey.* Distribute the tool across a broad range of projects and collect data later on the impact that the tool had on those projects.
- *Synthetic.* Have 20 programmers sitting at workstations spend two hours trying to debug a module, half using the tool and half using other techniques in order to quantify the differences that the tool provides.
- *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 and another tool to determine effectiveness in finding errors in a given module.
- *Static analysis.* Use a control flow analysis tool to see if one design method results in fewer logic errors than another design method.

In fact, for just about any technology, a data collection method can be devised to collect relevant data on that technology that conforms to any one of the twelve given data collection methods.

3 Classification of Experimental Models

We can summarize the previous discussions by Table 1. Most of the entries should be fairly obvious, although a few deserve some explanation.

A literature search can be replicated by others and should yield similar results. Static analysis can be either passive if it is an analysis of a completed system, or active, if the results of the analysis are used to direct development (e.g., limiting cyclomatic complexity of a given module to 10 or less). The same can be said of dynamic analysis.

We have to be very careful here; the example of cyclomatic complexity given above may have a dual purpose. If limiting cyclomatic complexity to 10 is part of the development method, then it is not an artifact of the experimental design, in which case the analysis is passive. However, if computing cyclomatic complexity is not part of the normal development plan, and if a manager

Exp. Model	Replicate	Local control	Factorial design	Influence	Temporal
Observational					
Proj. mon.	No	None	No	Passive	Current
Case study	No	Some	No	Active	Current
Assertion	No	Some	No	Active	Current
Survey	Yes	Some	No	Mostly passive	Current
Historical					
Lit. search	Yes	None	No	Passive	Past
Legacy data	No	None	No	Passive	Past
Lessons learned	No	None	No	Passive	Past
Stat. anal.	No	None	No	Both	Past or Current
Controlled					
Replicated	Yes	Some	Yes	Active	Current
Synthetic	Yes	Much	Yes	Active	Current
Dyn. anal.	No	Some	No	Active	Current
Simulation	Yes	Some	No	Active	Current

Table 1: Data collection experimental design.

decides to use those numbers to modify test plans, then the computation of the value does become an active part of the development cycle.

There are other relationships we can make among some of these techniques:

1. Project monitoring occurs during the lifetime of a project, while legacy data is a similar process that occurs after a project is completed.
2. A case study is a more intrusive version of a survey, which is the reason one is classified as passive and one as active.
3. A replicated experiment is often multiple instances of a case study.
4. Static analysis is a legacy data method that is solely concerned with the artifacts produced and not on the process of producing the artifacts.

Basili [6] has placed several of these experimental methods within a consistent framework (Table 2). An experiment can be *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

	Projects	
Teams	**1**	**Many**
1	in vivo (Case study)	in vivo (Survey)
Many	in vitro (Synthetic); in vivo (SEL studies)	in vivo (Replicated)

Table 2: Experimental models.

hand, is often a multiple team blocked (multiple individuals in most cases) in vitro study involving one project. Replicating this several times would be an in vitro blocked team multiple project study. Data collected by the NASA SEL could be considered a multiple team single project in vitro experiment. A survey could be classified as a vitro single team multiple project study. The interesting case of multiple team multiple project in vivo experiment is unlikely to occur in practice due to the high cost of replicating such experiments. Basili's model does not include our product or legacy methods, since they are not properly experimental designs.

Kitchenham [11] has another way to classify experimental design. She has grouped experiments under nine different classifications. A *quantitative experiment* is used to identify measurable benefits of using a method or tool; these may include measures like reduced production time, reduced rework, lower costs. A *qualitative experiment* is used to assess the features provided by a method or tool (e.g., the usability and effectiveness of a required feature, training requirements). The assessment results are based on use and subjective opinion.

There are three quantitative evaluations for determining explicit information about a method or tool:

1. A *quantitative experiment* is a formal experiment to evaluate the impact of a method or tool.
2. A *quantitative case study* is a case study to discover the impact of a method or tool.
3. A *quantitative survey* provides a quantitative survey to discover the impact of a method or tool.

There are also five qualitative evaluations of a method or tool:

1. A *qualitative screening* is performed by a single individual and may be a literature search to observe the feasibility of using a method or tool.
2. A *qualitative effects analysis* is a subjective assessment of a new technology based upon expert opinion. It can be viewed as a multiperson qualitative screening.
3. A *qualitative experiment* is a feature by feature evaluation by a group of users who will try a technology (e.g., method or tool) before making an evaluation.

	Quantitative	Qualitative
Screening	Not applicable	Lit. Search
Effects anal.	Not applicable	
Experiment	Synthetic, Replicated	Project monitoring
Case study	Case study	Legacy data, Lessons learned
Survey	Survey	

Benchmarking	Stat, anal., Dyn. anal., Simulation

Table 3: Kitchenham classifications.

4. A *qualitative case study* is a feature-based evaluation by someone who has used the new technology on a real project.

5. A *qualitative survey* is the opinion from a group of experts on the effectiveness of a new technology. It is related to a qualitative screening, but may be more biased in its conclusions since participation in the survey results would be voluntary on the part of the experts.

The final method is *benchmarking* where a number of standard tests are run against alternative technologies in order to assess their relative performance.

In Table 3 we list our set of data collection approaches and compare them to the nine classifications in the Kitchenham model. (Note that we classify project monitoring as a qualitative method since the experimenter has little control over the collection of necessary data.)

4 Model Validation

In order to test whether the classification presented here reflects the software engineering community's idea of experimental design and data collection, we looked at software engineering publications covering three different years: 1985, 1990, and 1995. In particular, 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. We classified each paper according to the data collection method used to validate the claims in the paper. We added the following two classifications in addition to the ones presented earlier:

1. *Not applicable.* Some papers did not address some new technology, so the concept of data collection does not apply. For example, a November, 1985 paper in *IEEE Software* described the goals of the then-new Carnegie Mellon University Software Engineering Institute [2]. It was not expected to have experimental data in it since it described a new organization.

2. *No experiment.* Some papers describing a new technology contained no experimental validation in it. Note that 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.

An appropriate research paper typically contains four sections [9]:

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 [9] observed that most papers contained some form of the first three sections, and the fourth evaluative phase was often missing. Tichy [20] performed a comprehensive study of 400 published papers, and arrived at a similar conclusion.

In our own 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 the 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 their 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 phase.

Before collecting this data, some of our hypotheses about this collection of papers were:

- Our model is complete; all papers should fall into one of our proposed classifications. Anecdotal evidence and studies such as the Tichy study would suggest, however, that *No experiment*, *Case study*, and *Assertion* would greatly outnumber the other methods.
- We wanted to see how well our data collection models agreed with the experimental models of the Tichy study. In particular, since there is growing interest in data collection and experimentation, we wanted to see if the percentage of papers that use some experimental validation increased between 1985 and 1990, and between 1990 and 1995.
- We would hope that replicated experiments would become more prevalent over time and that industrial experimentation would increase along with university data collection.

	ICSE15		ICSE18	
Method	**Count**	**%***	**Count**	**%**
Not applicable	1	–	6	–
No experimentation	16	34.0	20	37.7
Replicated	0		2	3.8
Synthetic	1	2.1	1	1.9
Dynamic analysis	0		2	3.8
Simulation	1	2.1	1	1.9
Project monitoring	0		0	
Case study	8	17.0	9	17.0
Assertion	6	12.8	4	7.5
Survey	0		1	1.9
Literature search	3	6.4	3	5.7
Legacy data	5	10.6	5	9.4
Lessons learned	4	8.5	5	9.4
Static analysis	3	6.4	0	

* - Percentages do not include "Not applicable" category.

Table 4: Pilot study – 1993 and 1996 conferences.

4.1 Pilot study

We first calibrated our model by each of us classifying the papers in the proceedings from the 1993 (ICSE 15) and 1996 (ICSE 18) conferences. The results are given in Table 4. In our initial test we had over an 80% agreement, which we jointly resolved, in classifying papers and believe now that we would agree on 95%+ of the time. It seems clear that one cannot get 100% agreement since "intent" of the experimenter often colors which of the 12 techniques is actually being applied. From this initial pilot study, about one third of all ICSE papers (for these two years) have no experimental validation.

4.2 Full study

The raw data for the complete study involved classification of 612 papers that were published in 1985, 1990, and 1995. This data is presented in Table 5 (ACM/IEEE ICSE conferences), Table 6 (IEEE Software Magazine), Table 7 (IEEE Transactions on Software Engineering), and Table 8 (Summary totals).

Method	**1985**	%	**1990**	%	**1995**	%	**Total**	%
Not applicable	6	–	1	–	5	–	12	–
No experimentation	16	32.0	12	35.3	10	37.0	38	34.2
Replicated	1	2.0	0		1	3.7	2	1.8
Synthetic	3	6.0	0		0		3	2.7
Dynamic analysis	0		0		0		0	
Simulation	2	4.0	0		1	3.7	3	2.7
Project monitoring	0		0		0		0	
Case study	5	10.0	7	20.6	4	14.8	16	14.4
Assertion	12	24.0	12	35.3	4	14.8	28	25.2
Survey	1	2.0	0		1	3.7	2	1.8
Literature search	1	2.0	1	2.9	0		2	1.8
Legacy data	1	2.0	2	5.9	1	3.7	4	3.6
Lessons learned	7	14.0	0		5	18.5	12	10.8
Static analysis	1	2.0	0		0		1	0.9

Table 5: ICSE Conferences

Method	**1985**	%	**1990**	%	**1995**	%	**Total**	%
Not applicable	2	–	3	–	5	–	10	–
No experimentation	15	39.5	21	36.8	5	13.2	41	30.8
Replicated	0		0		0		0	
Synthetic	1	2.6	1	1.2	0		2	1.5
Dynamic analysis	0		0		0		0	
Simulation	0		0		1	2.6	1	0.8
Project monitoring	0		1	1.8	0		1	0.8
Case study	2	5.3	6	10.5	6	15.8	14	10.5
Assertion	13	34.2	19	33.3	14	36.8	46	34.6
Survey	0		0		1	2.6	1	0.8
Literature search	1	2.6	5	8.8	3	7.9	9	6.8
Legacy data	1	2.6	0		1	2.6	2	1.5
Lessons learned	5	13.2	4	7.0	7	18.4	16	12.0
Static analysis	0		0		0		0	

Table 6: IEEE Software Magazine.

Method	1985	%	1990	%	1995	%	Total	%
Not applicable	0	–	1	–	0	–	0	–
No experimentation	60	40.8	42	34.7	16	20.8	118	34.2
Replicated	0		1	0.8	3	3.9	4	1.2
Synthetic	1	0.7	4	3.3	2	2.6	7	2.0
Dynamic analysis	0		3	2.5	4	5.2	7	2.0
Simulation	10	6.8	11	9.1	6	7.8	27	7.8
Project monitoring	0		0		0		0	
Case study	11	7.5	6	5	10	13.	27	7.8
Assertion	54	36.7	42	34.7	22	28.6	121	34.2
Survey	1	0.7	1	0.8	1	1.3	3	0.9
Literature search	3	2.	1	0.8	2	2.6	6	1.7
Legacy data	2	1.4	2	1.7	1	1.3	5	1.4
Lessons learned	4	2.7	8	6.6	8	10.4	20	5.8
Static analysis	1	0.7	0		2	2.6	3	0.9

Table 7: IEEE Transactions on Software Engineering.

Method	1985	%	1990	%	1995	%	Total	%
Not applicable	8	–	5	–	10	–	23	–
No experimentation	91	38.7	75	35.4	31	21.8	197	33.4
Replicated	1	0.4	1	0.5	4	2.8	6	1.0
Synthetic	5	2.1	5	2.4	2	1.4	12	2.0
Dynamic analysis	0		3	1.4	4	2.8	7	1.2
Simulation	12	5.1	11	5.2	8	5.6	31	5.3
Project monitoring	0		1	0.5	0		1	0.2
Case study	18	7.7	19	9.0	20	14.1	57	9.7
Assertion	79	33.6	73	34.4	40	28.2	192	32.6
Survey	2	0.9	1	0.5	3	2.1	6	1.0
Literature search	5	2.1	7	3.3	5	3.5	17	2.9
Legacy data	4	1.7	4	1.9	3	2.1	11	1.9
Lessons learned	16	6.8	12	5.7	20	14.1	48	8.1
Static analysis	2	0.9	0		2	1.4	4	0.7
Yearly totals	243		217		152		612	

Table 8: ICSE, Software, and TSE Summary Table.

4.2.1 Quantitative Observations

The most prevalent validation mechanisms appear to be lessons learned and case studies, each at a level of just under 10% (from Table 8). 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.

Much like in our pilot study, about one-third of the papers had no experimental validation; however, the percentages dropped from 38.7% in 1985 to 35.4% in 1990 to only 21.8% in 1995. Improvement in this important category seems to be occurring.

Tichy, in his study, 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.

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 Tichy's 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 with the weak form of assertion validation, we found that almost two-thirds of the papers did not have strong statistical validation of their reported claims. However a claim that 66% 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.

4.2.2 Qualitative Observations

We also offer the following observations on our classification of the 612 papers. It often was extremely difficult to classify individual papers. This was due to several major problems with many papers:

1. Authors often fail to clearly state exactly what their paper is about and exactly what their contribution to the software engineering literature is to be. Its hard to classify the validation if one doesn't know what is being validated. Authors often fail to clearly state their research hypothesis.

2. Authors fail to state how they propose to validate their hypotheses. We had to inspect each paper carefully in order 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 heading was not present and we had to determine if the data so presented could be called a validation.

3. Terms are used very loosely. Authors would use the term "case study" in a very informal manner, and even words like "controlled experiment" or "lessons learned" were used indiscriminantly. We attempted to classify each paper by what the authors actually did, not how they specified what they did. It is our hope that our paper can have some effect on formalizing these terms somewhat.

There are two caveats, however, in understanding the data presented in this paper:

1. Most of the 1985, 1990, and 1995 papers were classified by someone other than the authors of this paper who did the pilot study. Numerous spot checks were made on the classifications. The major difference seemed to be the classification of many of the case study papers as assertions. As we stated earlier, these two mechanisms are very similar, and it is a subjective opinion, to some extent, of in which category such papers are classified.
2. The papers that appear in a publication can be influenced greatly by the editor of that publication or program committee in the case of conferences. In our study, the editors and program committees from 1985, 1990, and 1995 were all different. This then imposes a confounding factor in our analysis process 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.

5 Conclusion

In a 1992 report from the National Research Council [15], the Panel on Statistical Issues and Opportunities for Research in the Combination of Information recommended:

> The panel urges 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 ...

Such problems are well-known in the software engineering world, and surveys such as the Tichy survey [20] and our own tend to validate the conclusion that the software engineering community can do a better job in reporting its results.

Toward that end, in this paper we have addressed the need to collect both product and process data in order to appropriately understand the development of software. We have developed a classification model that divides data collection activities into twelve passive and active methods. These twelve methods are grouped into three major categories of observational methods which look at contemporary data collection, historical data collection of completed projects, and controlled methods, which apply the scientific method in a controlled setting. This model includes the typical forms of experimentation that has been employed previously in software engineering in the past.

Via our analysis of some 600 published papers from 1985, 1990, and 1995, we observed that:

1. Too many papers have no experimental validation at all (about one-third), but fortunately, this number seems to be dropping.
2. Too many papers use an informal (assertion) form of validation. Better experimental design needs to be developed and used.
3. Lessons learned and case studies each are used about 10% of the time, the other techniques are used only a few percent at most.
4. Terminology of how one experiments is sloppy. We hope a classification model, such as ours, can help to encourage more precision in the describing of empirical research.

Our next step is to understand the relationship between a given method and the data that can be achieved by using it. Given a specific new technology, we need to understand which method best addresses the collection of data that can be used to validate the new technology.

6 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] Adrion W. R., Research methodology in software engineering, *Summary of the Dagstuhl Workshop on Future Directions in Software Engineering*, W. Tichy (Ed.) *ACM SIGSOFT Software Engineering Notes*, 18, 1, (1993).

[2] Barbacci M., N. Habermann, and M. Shaw, The Software Engineering Institute: Bridging Practice and Potential, *IEEE Software*, 2, 6 (November, 1985) 4-21.

[3] Basili V. R. and S. Green, Software Process Evolution at the SEL, *IEEE Software,* 11, 4, (July 1994) 58-66.

[4] 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.

[5] 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.

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

[7] Davis A., Eras of software technology transfer, *IEEE Software* 13, 2 (March, 1996) 4-7.

[8] Dijkstra E. W., *A Discipline of Programming*, Prentice Hall, 1976.

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

[10] International Standard ISO/IEC 12207-1, Information Technology – Software – Part 1; Software Life Cycle Processes, 1995.

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

[12] Leveson N. G., High-Pressure Steam Engines and Computer Software, *IEEE Computer* Vol. 27, No. 10 (1994) 65-73.

[13] Li N. R. and M. V. Zelkowitz, An Information Model for Use in Software Management Estimation and Prediction, Second International Conference on Information and Knowledge Management, Washington, DC, November, 1993, 481-489.

[14] Mills H. D, M. Dyer and R. C. Linger, Cleanroom Software Engineering, *IEEE Software*, Vol. 4, No. 5, (1987), pp. 19-25.

[15] 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.

[16] National Research Council, *Statistical Software Engineering*, Comm. on Applied and Theoretical Statistics, National Academy Press, Washington, DC, 1996.

[17] 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.

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

[19] Pratt T. and M. Zelkowitz, *Programming Languages: Design and Implementation*, Third Edition, Prentice Hall, (1996).

[20] 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.

[21] Zelkowitz M. V., Yeh R. T., Hamlet R. G., Gannon J. D., Basili V. R., Software engineering practices in the United States and Japan, *IEEE Computer* 17, 6 (1984) 57-66.