Annals of Software Engineering 1(1995)219–253

Experimental design and analysis in software engineering

Shari Lawrence Pfleeger†

Centre for Software Reliability, City University, Northampton Square, London EC1V 0HB, England

The paper presents key activities necessary for designing and analyzing an experiment in software engineering. After explaining how to choose an appropriate research technique to fit project goals, the paper shows how to state a hypothesis and determine how much control is needed over the variables involved. If control is not possible, then a formal experiment is not possible; a case study may be a better approach. Next, the six stages of an experiment (conception, design, preparation, execution, analysis and dissemination) are described, with design examined in detail. Design considerations such as replication, randomization and local control are discussed, and design techniques such as crossing and nesting are explained. Finally, data analysis is shown to be a function both of the experimental design and the distribution of the data. Throughout, examples are given to show how the techniques are interpreted and used in software engineering.

1. INTRODUCTION

Software engineers have many questions to answer. The test team wants to know which technique is best for finding faults in code. The maintainers seek the best tool to support configuration management. Project managers try to determine which types of experience make the best programmers or designers, while designers look for models that are good at predicting reliability. To answer these and other questions, we often rely on the advice and experience of others, which is not always based on careful, scientific research [Fenton *et al.* 1994]. As a software practitioner, it is important to make key decisions or assessments in an objective and scientific way. Even when we are not conducting our own research, we must be able to understand and evaluate the research of others. So we need to know two things: what assessment techniques are available to us, and which should be used in a given situation?

Suppose you are a software practitioner trying to evaluate a technique, method or tool. You can use two main types of assessment to answer your questions: case studies, surveys and formal experiments. Surveys are usually performed after the fact, and they try to capture the results of having used a method or tool. There is little

†E-mail: shari@csr.city.ac.uk

control over who used which method or tool. On the other hand, case studies and experiments are designed before the method or tool is used, and there is some degree of control over who uses which and when. This paper will introduce you to case studies and formal experiments, explaining how to decide which is appropriate for the investigation you want to undertake. Then, it focuses on key issues in experimental design, enabling you to understand how the design supports your decision-making. That is, the paper will help you to determine whether an experiment is appropriate for your evaluation, to design the experiment, and to derive meaningful results from it. The paper cannot design the specific experiment for you, but it can provide guidelines about what to consider in designing your own experiment. The techniques described and the situations used as examples are limited to those most likely to aid software development and maintenance decisions. Thus, there are experimental designs and situations not addressed; in those cases, you are referred to more general texts on statistics and experimental design for the broader, more generally applicable information.

Section 2 begins by exploring when experiments are appropriate, as opposed to case studies or other research techniques such as feature analysis. Then, assuming you have chosen to do an experiment, section 3 examines the steps required to perform an experiment. This section also describes in detail the principles of experimental design and gives examples of designs likely to be needed in software engineering research. Section 4 explains the analysis techniques required to evaluate the results of a formal experiment. Finally, section 5 summarizes the paper and recommends steps to be taken in evaluating the research of others.

2. CHOOSING A RESEARCH TECHNIQUE

Suppose your managers or colleagues have proposed the use of a new tool, technique or method. You have decided to investigate the tool, technique or method in a scientific way, rather than rely on "common wisdom". Your scientific investigation is likely to involve either a case study or a formal experiment. This section addresses the differences between case studies and formal experiments, giving examples of situations in which an experiment might be appropriate.

2.1 Stating the Hypothesis

The first step in your investigation is deciding what you want to investigate. That goal helps you to decide which type of research technique is most appropriate for your situation. The goal for your research can be expressed as a hypothesis you want to test. That is, you must specify what it is that you want to know or investigate. The **hypothesis** is the tentative theory or supposition that you think explains the behavior you want to explore. For example, your hypothesis may be "Using the ABC design method produces better quality software than using the XYZ design method".

Whether you evaluate a "snapshot" of your organization using ABC (a case study) or do a carefully controlled comparison of those using ABC with those using XYZ (a formal experiment), you are testing to see if the data you collect will confirm or refute the hypothesis you have stated. Thus, wherever possible, you should try to state your hypothesis in quantifiable terms, so that it is easy to tell whether the hypothesis is confirmed or refuted. For instance, rather than saying "Using the ABC design method produces better quality software than using the XYZ method", you can define "quality" in terms of the defects found and restate the hypothesis as "Code produced using the ABC design method has a lower number of defects per thousand lines of code than code produced using the XYZ method".

It is important to recognize that quantifying the hypothesis often leads to the use of surrogate measures. That is, in order to identify a quantity with a factor or aspect you want to measure (e.g. quality), you must measure the factor indirectly using something associated with that factor (e.g. defects). Because the surrogate is an indirect measure, there is danger that a change in the surrogate is not the same as a change in the original factor. For example, defects (or lack thereof) may not accurately reflect the quality or the software: finding a large number of defects during testing may mean that testing was very thorough and the resulting product is nearly defect-free, or it may mean that development was sloppy and there are likely to be many more defects left in the product. Similarly, delivered lines of code may not accurately reflect the amount of effort required to complete the product, since that measure does not take into account issues like reuse or prototyping. Therefore, along with a quantifiable hypothesis, you should document the relationship between the measures and the factors they intend to reflect. In particular, you should strive for quantitative terms that are as direct and unambiguous as possible.

2.2 Control over Variables

Once you have an explicit hypothesis, you must decide which variables can affect its truth, and how much control you have over each variable. The key discriminator between experiments and case studies is the degree of control over behavioral events and the variables they present. A case study is preferable when you are examining events where relevant behaviors cannot be manipulated. For example, if you are investigating the effect of a design method on the quality of the resulting software, but you have no control over who is using which design method, then you probably want to do a case study to document the results. Experiments are done when you can manipulate behavior directly, precisely and systematically. Thus, in the same example, if you can control who uses the ABC method, who uses XYZ, and when and where they are used, then an experiment is possible. This type of manipulation can be done in a "toy" situation, where events are organized to simulate their appearance in the real world, or in a "field" situation, where events are monitored as they actually happen.

This difference between case studies and formal experiments can be stated more rigorously by considering the notion of a **state variable**. A state variable is a factor that can characterize your project and influence your evaluation results. Examples of state variables include the application area, the system type, or the developers' experience with the language, tool or method. In a formal experiment you identify the variables and sample *over* them. This means that you select projects exhibiting a variety of characteristics typical for your organization and design your research so that more than one value will be taken for each characteristic. For example, your hypothesis may involve the effect of language on the quality of the resulting code. "Language" is a state variable, and an experiment would involve projects where many different languages would be used. You would design your experiment so that the projects involved represent as many languages as possible. In a case study, you sample *from* the state variable, rather than over it. This means that you select a value of the variable that is typical for your organization and its projects. With the language example, a case study might involve choosing a language that is usually used on most of your projects, rather than trying to choose a set of projects to cover as many languages as possible.

Thus, a state variable is used to distinguish the "control" situation from the "experimental" one in a formal experiment. (When you cannot differentiate control from experiment, you must do a case study instead of a formal experiment.) You may consider your current situation to be the control, and your new situation to be the experimental one; a state variable tells you how the experiment differs from the control. For example, suppose you want to determine whether a change in programming language can affect the productivity of your project. Then language is a state variable. If you currently use FORTRAN to write your programs and you want to investigate the effects of changing to Ada, then you can designate FORTRAN to be the control language and Ada to be the experimental one. The values of all other state variables should stay the same (e.g. application experience, programming environment, type of problem, and so on), so that you can be sure that any difference in productivity is attributable to the change in language.

2.3 Uses of Formal Experiments

There are many areas of software engineering that can be analyzed using formal experiments. One key motivator for using a formal experiment rather than a case study is that the results of an experiment are usually more generalizable than those of a case study. That is, if you use a case study to understand what is happening in a certain organization, the results apply only to that organization (and perhaps to organizations that are very similar). But because a formal experiment is carefully controlled and contrasts different values of the controlled variables, its results are generally applicable to a wider community and across many organizations. In the examples below, we will see how formal experiments can help answer a variety of questions.

2.3.1 Confirming Theories and "Conventional Wisdom"

Many techniques and methods are used in software engineering because "conventional wisdom" suggests that they are the best approaches. Indeed, many corporate, national and international standards are based on conventional wisdom. For example, many organizations use standard limits on McCabe's cyclomatic complexity measure or rules of thumb about module size to "assure" the quality of their software. However, there is very little quantitative evidence to support claims of effectiveness or utility of these and many other standards, methods or tools. Case studies can be used to confirm the truth of these claims in a single organization, while formal experiments can investigate the situations in which the claims are true. Thus, formal experiments can be used to provide a context in which certain standards, methods and tools are recommended for use.

2.3.2 Exploring Relationships

Software practitioners are interested in the relationships among various attributes of resources and software products. For example,

- How does the project team's experience with the application area affect the quality of the resulting code?
- How does the requirements quality affect the productivity of the designers?
- How does the design complexity affect the maintainability of the code?

Understanding these relationships is crucial to the success of any project. Each relationship can be expressed as a hypothesis, and a formal experiment can be designed to test the degree to which the relationship holds. Usually, many factors are kept constant or under control, and the value of one attribute is carefully varied to observe the effects of the changes. As part of the experiment's design, the attributes are clearly defined, consistently used and carefully measured.

For example, suppose you want to explore the relationship between programming language and productivity. Your hypothesis may state that certain types of programming languages (e.g. object-oriented languages) make programmers more productive than other types (e.g. procedural languages). A careful experiment would involve measuring not only the type of language and the resulting productivity, but also controlling other variables that might affect the outcome. That is, a good experimental design would ensure that factors such as programmer experience, application type, or development environment were controlled so that they would not confuse the results. After analyzing the outcome, you would be able to conclude whether or not programming language affects programmer productivity.

2.3.3 Evaluating the Accuracy of Models

Models are often used to predict the outcome of an activity or to guide the use of a method or tool. For example, size models such as function points suggest how

large the code may be, and cost models predict how much the development or maintenance effort is likely to cost. Capability maturity models guide the use of techniques such as configuration management or the introduction of testing tools and methods. Formal experiments can confirm or refute the accuracy and dependability of these models and their generality by comparing the predictions with the actual values in a carefully controlled environment.

Models present a particularly difficult problem when designing an experiment because their predictions often affect the outcome of the experiment. That is, the predictions become goals, and the developers strive to meet the goal, intentionally or not. This effect is common when cost and schedule models are used, and project managers turn the predictions into targets for completion. For this reason, experiments evaluating models can be designed as "double-blind" experiments, where the participants do not know what the prediction is until after the experiment is done. On the other hand, some models, such as reliability models, do not influence the outcome, since reliability measured as mean time to failure cannot be evaluated until the software is ready for use in the field. Thus, the time between consecutive failures cannot be "managed" in the same way that project schedules and budgets are managed.

2.3.4 Validating Measures

Many software measures have been proposed to capture the value of a particular attribute. For example, several measures claim to measure the complexity of code. A measure is said to be **valid** if it reflects the characteristics of an attribute under differing conditions. Thus, suppose code module X has complexity measure C. The code is augmented with new code, and the resulting module X′ is now perceived to be much more difficult to understand. If the complexity measure is valid, then the complexity measure C′ of X′ should be larger than C. In general, an experiment can be conducted to test whether a given measure appropriately reflects changes in the attribute it is supposed to capture.

Validating measures is fraught with problems. Often, validation is performed by correlating one measure with another. But surrogate measures used in this correlation can mislead the evaluator as described above. It is very important to validate using a second measure which is itself a direct and valid measure of the factor it reflects. Such measures are not always available or easy to measure. Moreover, the measures used must conform to human notions of the factor being measured. For example, if system A is perceived to be more reliable than system B, then the measure of reliability of A should be larger than that for system B; that is, the perception of "more" should be preserved in the mathematics of the measure. This preservation of relationship means that the measure must be objective and subjective at the same time: objective in that it does not vary with the measurer, but subjective in that it reflects the experience of the measurer.

2.4 Factors to Consider when Choosing Formal Experiments

Several general guidelines can help you decide whether to perform a case study or a formal experiment. As stated before, the central factor is the level of control needed for a formal experiment. If you have a high level of control over the variables that can affect the truth of your hypothesis, then you can consider an experiment. If you do not have that control, an experiment is not possible; a case study is the preferred technique. But the level of control satisfies the technical concerns; you must also address the practical concerns of research. It may be possible but very difficult to control the variables, either because of the high cost of doing so, or the degree of risk involved. For example, safety-critical systems may entail a high degree of risk in experimentation, and a case study may be more feasible.

The other key aspect to consider is the degree to which you can replicate the basic situation you are investigating. For instance, suppose you want to investigate the effects of language on the resulting software. Can you develop the same project multiple times using a different language each time? If replication is not possible, then you cannot do a formal experiment. However, even when replication is possible, the cost of replication may be prohibitive.

Table 1 summarizes these concerns. The next section assumes that you have used this table and have decided that a formal experiment is the appropriate research technique for your problem.

Table 1

Factors relating to choice of research technique.

Factor	Experiments	Case Studies
Level of control	High	Low
Difficulty of control	Low	High
Level of replication	High	Low
Cost of replication	Low	High

3. PRINCIPLES OF FORMAL EXPERIMENTS

Formal experiments, like software development itself, require a great deal of care and planning if they are to provide meaningful, useful results. This section discusses the planning needed to define and run a formal experiment, including consideration of several key characteristics of the experiment.

3.1 Procedures for Performing Experiments

There are several steps to carrying out a formal experiment:

- conception
- design
- preparation
- execution
- analysis
- dissemination and decision-making

We discuss each of these steps in turn.

3.1.1 Conception

The first step is to decide what you want to learn more about, and define the goals of your experiment. The conception stage includes the type of analysis described in section 2 to ensure that a formal experiment is the most appropriate research technique for you to use. Next, you must state clearly and precisely the objective of your study. The objective may include showing that a particular method or tool is superior in some way to another method or tool. Alternatively, you may wish to investigate whether, for a particular method or tool, differences in environmental conditions or quality can affect the use or output of the method or tool. No matter what you choose as your objective, it must be stated so that it can be clearly evaluated at the end of the experiment. That is, it should be stated as a question you want to answer. Then, the next step is to design an experiment that will provide the answer.

3.1.2 Design

Once your objective is clearly stated, you must translate the objective into a formal hypothesis, as described in section 2. Often, there are two hypotheses described: the null hypothesis and the experimental (or alternative) hypothesis. The **null hypothesis** is the one that assumes that there is no difference between two treatments (that is, between two methods, tools, techniques, environments, or other conditions whose effects you are measuring) with respect to the dependent variable you are measuring (such as productivity, quality or cost). The **alternative hypothesis** posits that there is a significant difference between the two treatments. For example, suppose you want to find out if the cleanroom technique of developing software produces code of higher quality than your current development process. Your hypotheses might be formulated as follows:

> *Null hypothesis*: There is no difference in code quality between code produced using cleanroom and code produced using our current process.
>
> *Alternative hypothesis*: The quality of code produced using cleanroom is higher than the quality of code produced using our current process.

It is always easy to tell which hypothesis is to be the null hypothesis and which the alternative. The distinguishing characteristic involves statistical assumptions: the null

hypothesis is assumed to be true unless the data indicates otherwise. Thus, the experiment focuses on departures from the null hypothesis, rather than on departures from the alternative hypothesis. In this sense, "testing the hypothesis" means determining whether the data is convincing enough to reject the null hypothesis and accept the alternative as true.

Hypothesis definition is followed by the generation of a formal design to test the hypothesis. The experimental design is a complete plan for applying differing experimental conditions to your experimental subjects so that you can determine how the conditions affect the behavior or result of some activity. In particular, you want to plan how the application of these conditions will help you to test your hypothesis and answer your objective question.

To see why a formal plan or design is needed, consider the following objective for an experiment:

> We want to determine the effect of using the Ada language on the quality of the resulting code.

The problem as stated is far too general to be useful. You must ask specific questions, such as

1. How is quality to be measured?
2. How is the use of Ada to be measured?
3. What factors influence the characteristics to be analyzed? For example, will experience, tools, design techniques, or testing techniques make a difference?
4. Which of these factors will be studied in the investigation?
5. How many times should the experiment be performed, and under what conditions?
6. In what environment will the use of Ada be investigated?
7. How should the results be analyzed?
8. How large a difference in quality will be considered important?

These are just a few of the questions that must be answered before the experiment can begin.

There is formal terminology for describing the components of your experiment. This terminology encourages you to consider all aspects of the experiment, so that it is completely planned to ensure useful results. The new method or tool you wish to evaluate (compared with an existing or different method or tool) is called the **treatment**. You want to determine if the treatment is beneficial in certain circumstances. That is, you want to determine if the treatment produces results that are in some way different. For example, you may want to find out whether a new tool increases productivity compared with your existing tool and its productivity. Or, you may want to choose between two techniques, depending on their effect on the quality of the resulting product.

Your experiment will consist of a series of tests of your methods or tools, and the experimental design describes how these tests will be organized and run. In any individual test run, only one treatment is used. An individual test of this sort is sometimes called a **trial**, and the **experiment** is formally defined as the set of trials. Sometimes, your experiment can involve more than one treatment, and you want to compare and contrast the differing results from the different treatments. The **experimental objects** or **experimental units** are the objects to which the treatment is being applied. Thus, a development or maintenance project can be your experimental object, and aspects of the project's process or organization can be changed to affect the outcome. Or the experimental objects can be programs or modules, and different methods or tools can be used on those objects. For example, if you are investigating the degree to which a design-related treatment results in reusable code components, you may consider design components as the experimental objects.

At the same time, you must identify *who* is applying the treatment; these people are called the **experimental subjects**. The characteristics of the experimental subjects must be clearly defined, so that the effects of differences among subjects can be evaluated in terms of the observed results.

When you are comparing using the treatment to not using it, you must establish a **control object**, which is an object not using the treatment. The control provides a baseline of information that enables you to make comparisons. In a case study, the control is the environment in which the study is being run, and it already exists at the time the study begins. In a formal experiment, the control situation must be defined explicitly and carefully, so that all the differences between the control object and the experimental object are understood.

The **response variables** or **dependent variables** are those factors that are expected to change or differ as a result of applying the treatment. In the Ada example above, quality may be considered as several attributes: the number of defects per thousand lines of code, the number of failures per thousand hours of execution time, and the number of hours of staff time required to maintain the code after deployment, for example. Each of these is considered a response variable. In contrast, **state variables** or **independent variables** are those variables that may influence the application of a treatment and thus indirectly the result of the experiment. State variables usually describe characteristics of the developers, the products, or the processes used to produce or maintain the code. It is important to define and characterize state variables so that their impact on the response variables can be investigated. Moreover, state variables can be useful in defining the scope of the experiment and in choosing the projects that will participate. For example, use of a particular design technique may be a state variable. You may decide to limit the scope of the experiment only to those projects that use Ada for a program design language, rather than investigating Ada on projects using any design technique. Or you may choose to limit the experiment to projects in a particular application domain, rather than considering all possible projects. Finally, as we saw in section 2, state variables (and the

control you have over them) help to distinguish case studies from formal experiments.

The number of and relationships among subjects, objects and variables must be carefully described in the experimental plan. The more subjects, objects and variables, the more complex the experimental design becomes and often the more difficult the analysis. Thus, it is very important to invest a great deal of time and care in designing your experiment, rather than rush to administer trials and collect data. Section 3.2 addresses in more detail the types of issues that must be identified and planned in a formal experimental design. In many cases, the advice in this section should be supplemented with the advice of a statistician, especially when many subjects and objects are involved. Once the design is complete, you will know what experimental factors (that is, response and state variables) are involved, how many subjects will be needed, from what population they will be drawn, and to what conditions or treatments each subject will be exposed. In addition, if more than one treatment is involved, the order of presentation or exposure will be laid out. Criteria for measuring and judging effects will be defined, as well as the methods for obtaining the measures.

3.1.3 Preparation

Preparation involves readying the subjects for application of the treatment. For example, tools may be purchased, staff may be trained, or hardware may be configured in a certain way. Instructions must be written out or recorded properly. If possible, a dry run of the experiment on a small set of people may be useful, so that you are sure that the plan is complete and the instructions understandable.

3.1.4 Execution

Finally, the experiment can be executed. Following the steps laid out in the plan, and measuring attributes as prescribed by the plan, you apply the treatment to the experimental subjects. You must be careful that items are measured and treatments are applied consistently, so that comparison of results is sensible.

3.1.5 Analysis

The analysis phase has two parts. First, you must review all the measurements taken to make sure that they are valid and useful. You organize the measurements into sets of data that will be examined as part of the hypothesis-testing process. Second, you analyze the sets of data according to the statistical principles described in section 4. These statistical tests, when properly administered, tell you if the null hypothesis is supported or refuted by the results of the experiment. That is, the statistical analysis gives you an answer to the original question addressed by the research.

3.1.6 Dissemination and Decision-Making

At the end of the analysis phase, you will have reached a conclusion about how the different characteristics you examined affected the outcome. It is important to document both methods and conclusions in a way that will allow your colleagues to duplicate your experiment and confirm your conclusions in a similar setting. To that end, you must document all of the key aspects of the research: the objectives, the hypothesis, the experimental subjects and objects, the response and state variables, the treatments, and the resulting data. Any other relevant documentation should be included: instructions, tool or method characteristics (e.g. version, platform, vendor), training manuals and more. You must state your conclusions clearly, making sure to address any problems experienced during the running of the experiment. For example, if staff changed during project development, or if the tool was upgraded from one version to another, you must make note of that.

The experimental results may be used in three ways. First, you may use them to support decisions about how you will develop or maintain software in the future: which tools or methods you will use, and in what situations. Second, others may use your results to suggest changes to their development environment. The others are likely to replicate your experiment to confirm the results on their similar projects. Third, you and others may perform similar experiments with variations in experimental subjects or state variables. These new experiments will help you and others to understand how the results are affected by carefully controlled changes. For example, if your experiment demonstrates a positive change in quality by using Ada, others may test to see if the quality can be improved still further by using Ada in concert with a particular Ada-related tool or in a particular application domain.

3.2 Principles of Experimental Design

Useful results depend on careful, rigorous and complete experimental design. In this section, we examine the principles that you must consider in designing your experiment. Each principle addresses the need for simplicity and for maximizing information. Simple designs help to make the experiment practical, minimizing the use of time, money, personnel and experimental resources. An added benefit is that simple designs are easier to analyze (and thus are more economical) than complex designs. Maximizing information gives you as complete an understanding of your experimental conditions and results as possible, enabling you to generalize your results to the widest possible situations.

Involved in the experimental design are two important concepts: experimental units and experimental error. As noted above, an **experimental unit** is the experimental object to which a single treatment is applied. Usually, you apply the treatment more than once. At the very least, you apply it to the control group as well as at least one group that differs from the control by a state variable. In many cases, you apply the treatment many times to many groups. In each case, you examine the results to see

what the differences are in applying the treatments. However, even when you keep the conditions the same from one trial to another, the results can turn out to be slightly different. For example, you may be investigating the time it takes for a programmer to recognize faults in a program. You have seeded a collection of programs with a set of known faults, and you ask a programmer to find as many defects as possible. But the programmer may take more time today to find the same set of faults as he or she took yesterday. To what is this variation attributable? **Experimental error** describes the failure of two identically treated experimental units to yield identical results. The error can reflect a host of problems:

- errors of experimentation
- errors of observation
- errors of measurement
- the variation in experimental resources
- the combined effects of all extraneous factors that can influence the characteristics under study but which have not been singled out for attention in the investigation

Thus, in our example of timing the programmer while he or she finds faults, the differences may be due to such things as:

- the programmer's mind wandered during the experiment
- the timer measured elapsed time inexactly
- the programmer was distracted by loud noises from another room
- the programmer found the faults in a different sequence today than yesterday

The aim of a good experimental design is to control for as many variables as possible, both to minimize variability among participants and to minimize the effects of irrelevant variables (such as noise in the next room or the order of presentation of the experiment). Ideally, we would like to eliminate the effects of other variables so that only the effects of the independent variables are reflected in the values of the dependent variable. That is, we would like to eliminate experimental error. Realistically, this complete elimination is not always possible. Instead, we try to design the experiment so that the effects of irrelevant variables are distributed equally across all the experimental conditions, rather than allowing them to inflate artificially (or bias) the results of a particular condition. In fact, statistician like whenever possible to measure the extent of the variability under "normal circumstances".

The three major principles described below, replication, randomization and local control, address this problem of variability by giving us guidance on forming experimental units so as to minimize experimental error. We consider each of these in turn.

3.2.1 Replication

Replication is the repetition of the basic experiment. That is, replication involves repeating an experiment under identical conditions, rather than repeating measurements on the same experimental unit. This repetition is desirable for several reasons. First, replication (with associated statistical techniques) provides an estimate of experimental error that acts as a basis for assessing the importance of observed differences in an independent variable. That is, replication can help us to know how much confidence we can place in the results of the experiment. Second, replication enables us to estimate the mean effect of any experimental factor.

It is important to ensure that replication does not introduce the confounding of effects. Two or more variables are said to be **confounded** if it is impossible to separate their effects when the subsequent analysis is performed. For example, suppose you want to compare the use of a new tool with your existing tool. You set up an experiment where programmer A uses the new tool in your development environment, while programmer B uses the existing tool. When you compare measures of quality in the resulting code, you cannot say how much of the difference is due to the tools because you have not accounted for the difference in the skills of the programmers. That is, the effects of the tools (one variable) and the programmer's skills (another variable) are confounded. This confounding is introduced with the replication when the repetition of the experiment does not control for other variables (like programmer skills).

Similarly, consider the comparison of two testing techniques. A test team is trained in test technique X and asked to test a set of modules. The number of defects discovered is the chosen measure of the technique's effectiveness. Then, the test team is trained in test technique Y, after which they test the same modules. A comparison of the number of defects found with X and with Y may be confounded with the similarities between techniques or a learning curve in going from X to Y. Here, the sequence of the repetition is the source of the confounding.

For this reason, the experimental design must describe in detail the number and kinds of replications of the experiments. It must identify the conditions under which each experiment is run (including the order of experimentation), and the measures to be made for each replicate.

3.2.2 Randomization

Replication makes possible a statistical test of the significance of the results. But it does not ensure the validity of the results. That is, we want to be sure that the experimental results clearly follow from the treatments that were applied, rather than from other variables. Some aspects of the experimental design must organize the experimental trials in a way that distributes the observations independently, so that the results of the experiment are valid. **Randomization** is the random assignment of subjects to groups or of treatments to experimental units, so that we can assume

independence (and thus validity) of results. Randomization does not guarantee independence, but it allows us to assume that the correlation on any comparison of treatment is as small as possible. In other words, by randomly assigning treatments to experimental units, you can try to keep some treatment results from being biased by sources of variation over which you have no control.

For example, sometimes the results of an experimental trial can be affected by the time, the place or unknown characteristics of the participants. These uncontrollable factors can have effects that hide or skew the results of the controllable variables. To spread and diffuse the effects of these uncontrollable or unknown factors, you can assign the order of trials randomly, assign the participants to each trial randomly, or assign the location of each trial randomly, whenever possible.

A key aspect of randomization involves the assignment of subjects to groups and treatments. If we use the same subjects in all experimental conditions, we say that we have a related within-subjects design. However, if we use different subjects in different experimental conditions, we have an unrelated between-subjects design. If there is more than one independent variable in the experiment, we can consider the use of same or different subjects separately for each of the variables. (We will describe this issue in more detail later on.)

Thus, your experimental design should include details about how you plan to randomize assignment of subjects to groups or of treatments to experimental units. In cases where complete randomization is not possible, you should document the areas where lack of randomization may affect the validity of the results. In later sections, we shall see examples of different designs and how they involve randomization.

3.2.3 Local Control

As noted in section 2, one of the key factors that distinguishes a formal experiment from a case study is the degree of control. **Local control** is the aspect of the experimental design that reflects how much control you have over the placement of subjects in experimental units and the organization of those units. Whereas replication and randomization ensure a valid test of significance, local control makes the design more efficient by reducing the magnitude of the experimental error. Local control is usually discussed in terms of two characteristics of the design: blocking and balancing the units.

Blocking means allocating experimental units to blocks or groups so the units within a block are relatively homogeneous. The blocks are designed so that the predictable variation among units has been confounded with the effects of the blocks. That is, the experimental design captures the anticipated variation in the blocks by grouping like varieties, so that the variation does not contribute to the experimental error. For example, suppose you are investigating the comparative effects of three design techniques on the quality of the resulting code. The experiment involves teaching the techniques to twelve developers and measuring the number of defects

found per thousand lines of code to assess the code quality. It may be the case that the twelve developers graduated from three universities. It is possible that the universities trained developers in very different ways, so that the effect of being from a particular university can affect the way in which the design technique is understood or used. To eliminate this possibility, three blocks can be defined so that the first block contains all developers from university S, the second block from university Y, and the third block from university Z. Then, the treatments are assigned at random to the developers from each block. If the first block has six developers, you should expect two to be assigned to design method A, two to B, and two to C, for instance.

Balancing is the blocking and assignment of treatments so that an equal number of subjects is assigned to each treatment, wherever possible. Balancing is desirable because it simplifies the statistical analysis, but it is not necessary. Designs can range from being completely balanced to having little or no balance.

In experiments investigating only one factor, blocking and balancing play important roles. If the design includes no blocks, then it must be completely randomized. That is, subjects must be assigned at random to each treatment. A balanced design, with equal numbers of subjects per treatment, is preferable but not necessary. If one blocking factor is used, subjects are divided into blocks and then randomly assigned to each treatment. In such a design, called a *randomized block design*, balancing is essential for analysis. Thus, this type of design is sometimes called a *complete block design*. Sometimes, units are blocked with respect to two different variables (e.g. staff experience and program type) and then assigned at random to treatments so that each blocking variable combination is assigned to each treatment an equal number of times. In this case, called *Latin Square*, balancing is mandatory for correct analysis.

Your experimental design should include a description of the blocks defined and the allocation of treatments to each. This part of the design will assist the analyst in understanding which statistical techniques apply to the data that result from the experiments. More detail about the analysis can be found in section 4.

3.3 Types of Experimental Designs

There are many types of experimental designs. It is useful to know and understand the several types of designs that you are likely to use in software engineering research, since the type of design constrains the type of analysis that can be performed and therefore the types of conclusions that can be drawn. (A description of these analysis techniques can be found in [Kitchenham 1992].) For example, there are several ways to calculate the F-statistic for an analysis of variance; the choice of calculation depends on the experimental design, including the number of variables and the way in which the subjects are grouped and balanced. Similarly, the measurement scale of the variables constrains the analysis. Nominal scales simply divide data into categories and can be analyzed by using statistical tests such as the Sign test (which looks at the direction of a score or measurement); on the other hand, ordinal scales permit

rank ordering and can be investigated with more powerful tests such as Wilcoxon (looking at the size of the measurement differences). Parametric tests such as analysis of variance can be used only on data that is at least of an interval scale.

The sampling also enforces the design and constrains the analysis. For example, the amount of random variance should be equally distributed among the different experimental conditions if parametric tests are to be applied to the resulting data. Not only does the degree of randomization make a difference to the analysis, but also the distribution of the resulting data. If the experimental data is normally or near-normally distributed, then you can use parametric tests. However, if the data is not normally distributed, or if you do not know what the distribution is, nonparametric methods are preferable.

Many investigations involve more than one independent variable. In addition, the experiment invokes changes in the dependent variable as one or more of the independent variables changes. An independent variable is called a **factor** in the experimental design. For example, a study to determine the effects of experience and language on the productivity of programmers might have two factors: experience and language. The dependent variable is productivity. Various values or classifications for each factor are called the **levels** of the factor. Levels can be continuous or discrete, quantitative or qualitative. If experience is measured in years of experience as a programmer, then each integer number of years can be considered a level. If the most experienced programmer in the study has eight years of experience, and if there are five languages in the study, then the first factor has eight levels and the second factor five.

There are several types of factors, reflecting such things as treatments, replications, blocking and grouping. This document does not tell you what factors should be included in your design. Neither does it prescribe the number of factors nor the number of levels. Instead, it explains how the factors can be related to each other, and how the levels of one factor are combined with the levels of another factor to form the treatment combinations. The remainder of section 3 explains how to derive a design from the number of factors and levels you want to consider in your investigation.

Most designs in software engineering research are based on two simple relations between factors: crossing and nesting; each is discussed separately.

3.3.1 Crossing

The design of an experiment can be expressed in a notation that reflects the number of factors and how they relate to the different treatments. Expressing the design in terms of factors, called the **factorial design**, tells you how many different treatment combinations are required. Two factors, A and B, in a design are said to be **crossed** if each level of one factor appears with each level of the other factor. This relationship is denoted A × B. The design itself is illustrated in figure 1, where a_i represents the levels of factor A and b_j the levels of factor B. The figure's first row

Crossed		Factor B	
		Level 1	Level 2
A	1	a1 b1	a1 b2
	2	a2 b1	a2 b2

Figure 1. Example of crossed design.

Nested

Factor A			
Level 1		Level 2	
Factor B		Factor B	
Level 1	Level 2	Level 1	Level 2
a1 b1	a1 b2	a2 b1	a2 b2

Figure 2. Example of a nested design.

indicates that you must have a treatment for level 1 of A occurring with level 1 of B, and of level 1 of A occurring with level 2 of B. The first column shows that you must have a treatment for level 1 of B occurring with each of the two levels of A. In the previous example, the effects of language and experience on productivity can be written as an 8×5 crossed design, requiring 40 different treatment combinations. This design means that your experiment must include treatments for each possible combination of language and experience. For three factors, A, B and C, the design $A \times B \times C$ means that all combinations of all the levels occur.

3.3.2 Nesting

Factor B is **nested** within factor A if each meaningful level of B occurs in conjunction with only one level of factor A. The relationship is depicted as B(A), where B is the nested factor and A is the nest factor. For example, consider again the effects of language and experience on productivity. However, let factor A be the language, and B be the years of experience with a particular language. Now B is dependent on A, and each level of B occurs with only one level of A. That is, B is nested within A. A two-factor nested design is depicted in figure 2.

Nesting can involve more than two factors. For example, three factors can be nested as C(B(A)). In addition, more complex designs can be created as nesting and crossing are combined.

There are several advantages to expressing a design in terms of factorials. First, factorials ensure that resources are used most efficiently. Second, information obtained in the experiment is complete and reflects the various possible interactions among variables. Consequently, the experimental results and conclusions drawn from them are applicable over a wider range of conditions than they might otherwise be. Finally, the factorial design involves an implicit replication, yielding the related benefits in terms of reduced experimental error.

On the other hand, the preparation, administration and analysis of a complete factorial design is more complex and time-consuming than a simple comparison. With a large number of treatment combinations, the selection of homogeneous experimental units is difficult and can be costly. And some of the combinations may be impossible or of little interest to you, wasting valuable resources. For these reasons, the remainder of section 3 explains how to choose an appropriate experimental design for your situation.

3.4 Selecting an Experimental Design

As seen above, there are many choices for how to design your experiment. The ultimate choice depends on two things: the goals of your investigation and the availability of resources. The remainder of this section explains how to decide which design is right for your situation.

3.4.1 Choosing the Number of Factors

Many experiments involve only one variable factor. The experiments may be quite complex, in that there may be many levels of the variable that are compared (e.g. the effects of many types of languages, or of several different tools). One-variable experiments are relatively simple to analyze, since the effects of the single factor are isolated from other variables that may affect the outcome. However, it is not always possible to eliminate the effects of other variables. Instead, we strive to minimize the effects, or at least distribute the effects equally across all the possible

conditions we are examining. For example, techniques such as randomization aim to prevent variability among people from biasing the results.

However, sometimes the absence of a second variable affects performance in the first variable. That is, people act differently in different circumstances, and you may be interested in the variable interactions as well as in individual variables. For instance, suppose you are considering the effects of a new design tool on productivity. The design tool may be used differently by designers who are well-versed in object-oriented design from those who are new to object-oriented design. If you were to design a one-factor experiment by eliminating the effects of experience with object-oriented design, you would get an incomplete (and probably incorrect) view of the effects of the tool. It is better to design a two-factor experiment that incorporates both use of the tool and designer experience. That is, by looking at the effects of several independent variables, you can assess not only the individual effects of each variable (known as the main effect of each variable) but also any possible interactions among the variables.

To see what we mean by interaction, consider the reuse of existing code. Suppose your organization has a repository of code modules that is made available to some of the programmers but not all. You design an experiment to measure the time it takes to code a module, distinguishing small modules from large. When the experiment is complete, you plot the results, separating the times for those who

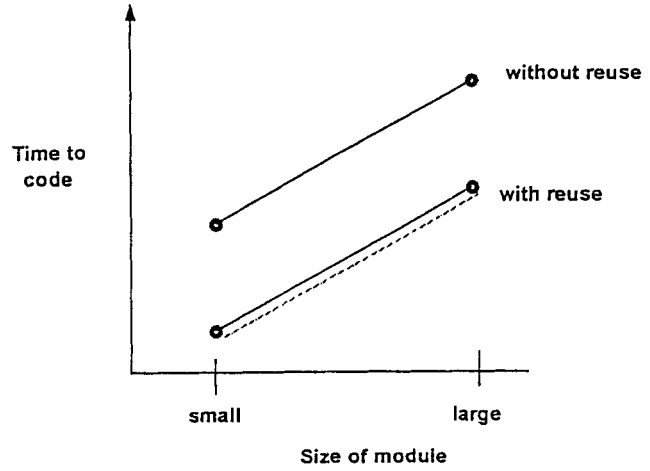


Figure 3. No interaction between factors.

reused code from the times of those who did not. This experiment has two factors: module size and reuse. Each factor has two levels; module size is either small or large, and reuse is either present or absent. If the results of your experiment resemble figure 3, then you can claim that there is no interaction between the two factors.

Notice that the lines in figure 3 are parallel: the parallelism is an indication that the two variables have no effects on each other. Thus, an individual line shows the main effect of the variable it represents. In this case, each line shows that the time to code a module increases with the size of the module. In comparing the two lines, we see that reuse reduces the time to code a module, but the parallel lines indicate that size and degree of reuse do not change the overall trend. However, if the results resemble figure 4, then there is indeed interaction between the variables, since the

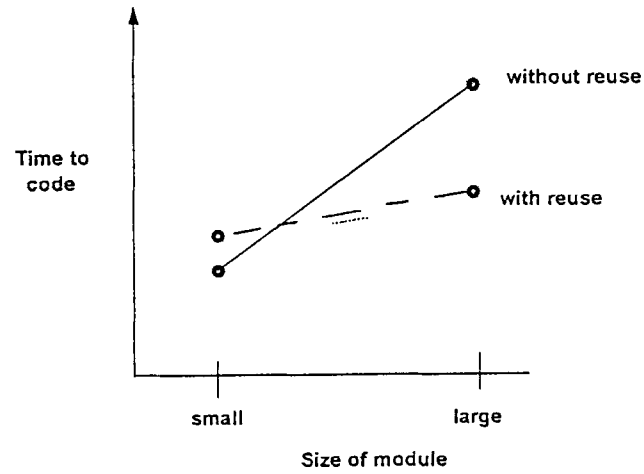


Figure 4. Interaction between factors.

lines are not parallel. Such a graph may result if there is considerable time spent searching through the repository. For a small module, it may actually take more time to scan the repository than to code the module from scratch. For large modules, reuse is better than writing the entire module, but there is still significant time eaten up in working with the repository. In other words, there is interaction between size of module and reuse of code.

Thus, there is far more information available from the two-factor experiment than there would have been from two one-factor experiments. The latter would have confirmed that the time to code increases with the size of the module, both with and without reuse. But the two-factor experiment shows the relationship between the factors as well as the single-factor results. In particular, it shows that, for small modules, reuse may not be warranted. In other words, multiple-factor experiments offer multiple views of the data and enlighten us in ways that are not evident from a collection of single-factor experiments.

Another way of thinking about whether to use one factor or more is to decide what kind of comparison you want to make. If you are examining a set of competing treatments, you can use a single-factor experiment. For example, you may want to

investigate three design methods and their effects on quality. That is, you apply design methods A, B and C to see which yields the highest quality design or code. Here, you do not have other variables that may interact with the design method.

On the other hand, you may want to investigate treatment combinations, rather than competing treatments. For example, instead of looking just at design methods, you want to analyze the effects of design methods in conjunction with tool assistance. Thus, you are comparing design method A with tool assistance to design method A without tool assistance, as well as design method B with tool assistance. You have a two-factor experiment: one factor is the design method, and the other is the absence or presence of tool assistance. Your experiment tests $n_1 + n_2$ different treatments, where n_1 is the number of levels in factor 1, and n_2 is the number of levels in factor 2.

3.4.2 Factors vs. Blocks

Once you decide on the number of factors appropriate for your experiment, you must determine how to use blocking to improve the experiment's precision. To see how to decide when something should be a block instead of a factor., we use the example of staff experience with software design.

In many experiments, we suspect that the experience of the subjects will affect the outcome. One option in the experimental design is to treat experience as a blocking factor, as described in section 3.2.3. To do this, we can assess the experience of the designers in terms of the number of years each has had experience with design. We can match staff with similar experience backgrounds and then assign staff randomly to the different treatments. Thus, if we are investigating design methods A and B, each block will have at least two different subjects of approximately equal experience; within each block, the subjects are assigned randomly to methods A and B.

On the other hand, if we treat experience as a factor, we must define levels of experience and assign subjects in each level randomly to the alternative levels of the other factor. In the design example, we can classify designers as having high and low experience (the two levels of experience); then, within each group, subjects are assigned at random to design method A or B.

To determine which approach (factor or block) is best, consider the basic hypothesis. If we are interested in whether design A is better than design B, then experience should be treated as a blocking variable. However, if we are interested in whether the results of using design methods A and B are influenced by staff experience, then experience should be treated as a factor. Thus, as we saw in section 3.4.1, if we are not interested in interactions, then blocking will suffice; if interactions are important, then multiple factors are needed.

In general, then, we offer the following guidelines about blocking:

> If you are deciding between two methods or tools, then you should identify state variables that are likely to affect the results and sample over those

variables using blocks to ensure an unbiased assignment of experimental units to the alternative methods or tools.

If you are deciding among methods or tools in a variety of circumstances, then you should identify state variables that define the different circumstances and treat each variable as a factor.

In other words, use blocks to eliminate bias; use factors to distinguish cases or circumstances.

3.4.3 *Choosing between Nested and Crossed Designs*

Once you have decided on the appropriate number of factors for your experiment, you must select a structure that supports the investigation and answers the questions you have. As we shall see, this decision is more complicated in software engineering than in other disciplines, because assigning a group not to use a factor may not be sensible or even possible. That is, there are hidden effects that must be made explicit, and there are built-in biases that must be addressed by the structure of the experiment. In addition, other issues can complicate this choice.

Suppose that a company wants to test the effectiveness of two design methods, A and B, on the quality of the resulting design, with and without tool support. The company identifies twelve projects to participate in the experiment. For this experiment, we have two factors: design method and tool usage. The first factor has two levels, A and B, and the second factor also has two levels, use of the tool and lack of use. A crossed design makes use of every possible treatment combination, and it would appear that a crossed design could be used for this experiment.

Crossed		Design Method	
		Method A	Method B
Tool Usage	Not used	Projects 1, 2 and 3	Projects 7, 8 and 9
	Used	Projects 4, 5 and 6	Projects 10, 11 and 12

Figure 5. Crossed design for design methods and tool usage.

As shown in figure 5, the twelve projects are organized so that three projects are assigned at random to each treatment in the experimental design. Consider the

implications of the design as shown. Any project has been assigned to any treatment. However, unless the tools used to support method A are exactly the same as the tools used to support method B, the factor levels for tool usage are not comparable within the two methods. In other words, with a crossed design such as this, we must be able to make sense of the analysis in terms of interaction effects. We should be able to investigate down columns (in this example, does tool usage make a difference for a given method?) as well as across rows (in this example, does method make a difference with the use of a given tool?). With the design in figure 5, the interaction between method and tool usage (across rows) is not really meaningful. The crossed design yields four different treatments based on method and tool usage that allow us to identify which treatment produces the best result. But the design does not allow us to make statements about the interaction between tool usage and method type.

We can remedy this situation by using a nested design, as shown in figure 6. The nested design is analyzed differently from the crossed design (a one-way analysis of variance, as opposed to a two-way analysis of variance), so there is no risk of meaningless interaction effects, as there was with the crossed design.

Design Method			
Method A		Method B	
Tool Usage		Tool Usage	
Not used	Used	Not used	Used
Projs. 1,2,3	Projs. 4,5,6	Projs. 7,8,9	Projs. 10,11,12

Figure 6. Nested design for design methods and tool usage.

Thus, a nested design is useful for investigating a factor with two or more conditions, while a crossed design is useful for looking at two factors, each with two or more conditions. This rule of thumb can be extended to situations with more than two factors. However, the more factors, the more complex the resulting analysis. For the remainder of this document, we focus on at most two factors, as most situations in software engineering research will involve only one or two factors, with blocking and randomization used to ameliorate the effects of other state variables.

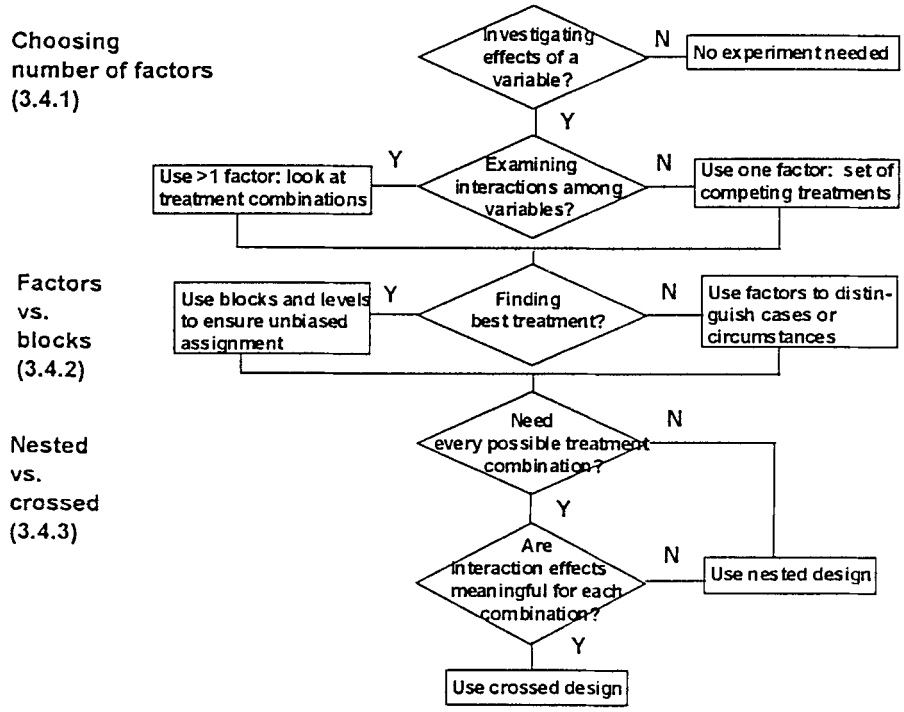


Figure 7. Flow chart for choosing design.

Figure 7 summarizes some of the considerations explained so far. Its flow chart helps you to decide on the number of factors, whether to use blocks, and whether to consider a crossed or nested design.

However, there are other, more subtle issues to consider when selecting a design. Let us examine two more examples to see what kinds of problems may be hidden in an experimental design. Consider first the crossed design described by figure 8. The design shows an experiment to investigate two factors: staff experience and design method type. There are two levels of experience, high and low, and two types of design method. The staff can be assigned to a project after the project's status is determined by a randomization procedure. Then, the project can be assigned to a treatment combination. This example illustrates the need to randomize in several ways, as well as the importance of assigning subjects and treatments in an order that makes sense to the design and the goals of the experiment.

Figure 9 is similar to figure 8, except that it is examining method usage, as opposed to method type. In this case, it is important to define exactly what is meant by "not used". Unlike medicine and agriculture, where "not used" means the use of a placebo or lack of treatment with a chemical, "not used" in software engineering may be difficult or impossible to control. If we tell designers not to use a particular

Crossed		Design Method	
		Method A	Method B
Staff Experience	Low	Projects 1, 2 and 3	Projects 7, 8 and 9
	High	Projects 4, 5 and 6	Projects 10, 11 and 12

Figure 8. Crossed design for method types and staff experience.

Crossed		Design Method	
		Used	Not used
Staff Experience	Low	Projects 1, 2 and 3	Projects 7, 8 and 9
	High	Projects 4, 5 and 6	Projects 10, 11 and 12

Figure 9. Crossed design for method usage and staff experience.

method, they are likely to use an alternative method, rather than no method at all. The alternative method may be hidden, based on how they were trained or what experience they have, rather than an explicitly-defined and well-documented other method. In this case, the experimental design is inappropriate for the goals of the experiment. However, if the goal of the experiment is to assess the benefit of a tool to support the given method, then the design is sufficient.

3.4.4 Fixed and Random Effects

Some factors allow us to have complete control over them. For example, we may be able to control which language is used to develop a system, or which processor the system is developed on. But other factors are not easy to control, or are pre-determined: staff experience is an example of this type of factor. The degree

of control over factor levels is an important consideration in choosing an experimental design. A fixed-effects model has factor levels or blocks that are controlled. A random-effects model has factor levels or blocks that are random samples from a population of values. If staff experiences is used as a blocking factor to match subjects of similar experience prior to assigning them to a treatment, then the actual blocks are a sample of all possible blocks, and we have a random-effects model. However, if staff experience is defined as two levels, low and high, the model is a fixed-effects model.

The difference between fixed- and random-effects models affects the way the resulting data is analyzed. For completely randomized experiments, there is no difference in analysis. But for more complex designs, the difference affects the statistical methods to assess the results. If you are not using a completely randomized experiment, you should consult a statistician to verify that you are planning to use techniques appropriate to the type of effects in your model.

The degree of randomization also affects the type of design that is used in your experiment. You can choose a crossed design when subjects can be assigned to all levels (for each factor) at random. For example, you may be comparing the use of a tool (in two levels: using the tool and not using the tool) with the use of a hardware platform (using a Sun, a PC or a Macintosh, for instance). Since you can assign developers to each level at random, your crossed design allows you to look for interaction between the tool and the platform. On the other hand, if you are comparing tool usage and experience (low level of experience versus high level of experience), then you cannot assign people at random to the experience; a nested design is more appropriate here.

3.4.5 Matched or Same Subject Designs

Sometimes, economy or reality prevents us from using different subjects for each type of treatment in our experimental design. For instance, we may not find enough programmers to participate in an experiment, or we do not have enough funds to pay for a very large experiment. We can use the same subjects for different treatments, or we can try to match subjects according to their characteristics in order to reduce the scale and cost of the experiments. For example, we can ask the same programmer to use tool A in one situation and then tool B in another situation. The design of matched- or same-subject experiments allows variation among staff to be assessed and accounts for the effects of staff differences in analysis. This type of design usually increases the precision of an experiment, but it complicates the analysis.

Thus, when designing your experiment, you should decide how many and what types of subjects you want to use. For experiments with one factor, you can consider testing the levels of the factor with the same subjects or with different subjects. For two or more variables, you can consider the question of same-or-different separately for each variable. To see how, suppose you have an experimental design with four

different treatments, generated by a crossed design with two factors. If different subjects are used for each treatment (that is, for each of both variables), then you have a completely unrelated between-subjects design. Alternatively, you could use the same subjects (or subjects matched for similar value of each level) and subject them to all four treatments; this is a completely related within-subjects design. Finally, you can use the same subjects for one factor but different subjects for the other factor to yield a mixed between- and within-subjects design.

3.4.6 Repeated Measurements

In many experiments, one measurement is made for each item of interest. However, it can be useful to repeat measurements in certain situations. For example, repeating a measurement can be useful in validating it, by assessing the error associated with the measurement process. We explain the added value of repeated measurements by describing an example.

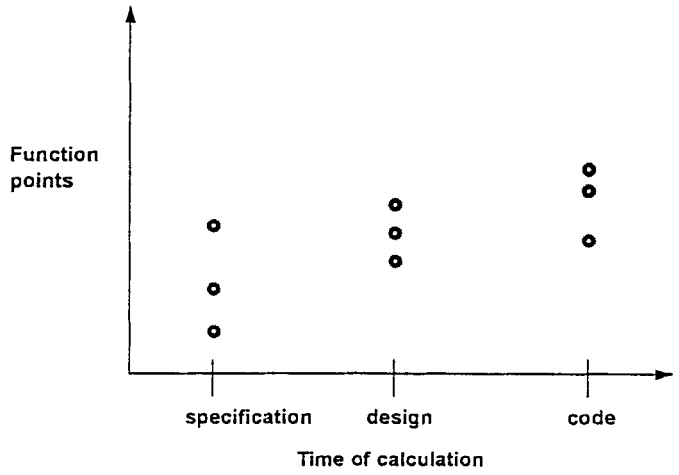


Figure 10. Repeated measurements on function point calculations.

Figure 10 depicts the results of an experiment involving one product and three developers. Each developer was asked to calculate the number of function points in the product at each of three different times during development: after the specification was completed, after the design was finished, and after the code was done. Thus, in the figure, there are three points marked at each of the three estimation times. For example, at specification, each of the three developers produced a slightly different function point estimate, so there are three distinct points indicated above "specification" on the x-axis. The figure shows that there were two kinds of variation in the data that resulted. The horizontal variation indicates the variation over time, while the vertical

differences at each measurement time indicates the variation due to the differences among the developers. Clearly, these repeated measurements add value to the results of the experiment, but at the cost of the more complex analysis required. The horizontal variation helps us to understand the error about the line connecting the means at each measurement time, and the vertical error helps us to understand observational error.

As you can see, there are many issues to consider when choosing a design for your experiment. Once it is chosen and the experiment is run, the resulting data can be analyzed. The next section explains how to select appropriate analysis techniques.

4. ANALYZING THE RESULTS OF EXPERIMENTS

After you have collected the relevant data, you must analyze it in an appropriate way. This section describes the items you must consider in choosing the analysis techniques. We describe typical situations for which you may be performing an experiment, and which technique is most appropriate for each situation. Specific statistical techniques are described and used in the discussion, but the details of each statistical approach (including formulae and references to other statistical textbooks) are found in [Kitchenham 1992; Caulcutt 1993; Chatfield 1993].

There are three major items to consider when choosing your analysis techniques: the nature of the data you collected, why you performed the experiments, and the type of experimental design you used. We consider each of these in turn.

4.1 Distribution of Data

The nature of your data will help you to decide which analysis techniques are available to you. Thus, it is very important for you to understand that the data are a sample from a larger population of all the data you could have gathered, given infinite resources. Because you do not have infinite resources, you are using your relatively small sample to generalize to that larger population, so the characteristics of the population are important. Many statistical techniques assume that the data is normally distributed, and your sample is randomly chosen from that larger distribution.

However, most software-related measurements are not normally distributed. Whereas a normal distribution is continuous and symmetric about its mean, much software data is discrete and not symmetric. In fact, it is unusual for software data to be a random sample from a well-defined population. For example, the distribution of software module size data can be very different from a normal distribution. You can tell if your data is normal by doing several tests. For instance, in a normal distribution, the three major indicators of central tendency (mean, median and mode) are the same. [Kitchenham 1992] describes several others. If you are not sure whether your data is normal, you must assume that it is not, and use techniques for evaluating non-normal data.

There are several ways to analyze data that is non-normal:

- Use robust statistics and non-parametric analysis methods.
- Transform your measurements to a scale (such as the logarithmic scale) that conforms more closely to a normal distribution.
- Determine the true underlying distribution and choose techniques appropriate to it.

In the examples that follow, we will consider both normal and non-normal cases, depending on the type of data.

4.2 Purpose of the Experiment

In section 2, we noted four major reasons to conduct a formal experiment:

- to confirm a theory
- to explore a relationship
- to evaluate the accuracy of a model
- to validate a measure

Each of these requires analysis carefully designed to meet the stated experimental objective. In particular, the objective is expressed formally in terms of the hypothesis, and the analysis must address the hypothesis directly.

4.2.1 Confirming a Theory

Your experiments may be designed to explore the truth of a theory. The theory usually states that use of a certain method, tool or technique (the treatment) has a particular effect on the experimental subjects, making it better in some way than another treatment (usually the existing method, tool or technique). For example, you may want to investigate the effect of the cleanroom technique by comparing it with your existing testing methods. The usual analysis approach for this situation is *analysis of variance*. That is, you consider two populations, the one that uses the old technique and the one that uses the new, and you do a statistical test to see if the difference in treatment results is statistically significant.

There are two cases to consider: normal data and non-normal data. If the data comes from a normal distribution and you are comparing two groups, you can use the *Student's t-test* to analyze the effects of the two treatments. If you have more than two groups to compare, a more general analysis of variance test, using the *F statistic*, is appropriate. Both of these are described in most statistics books.

For example, suppose you are investigating the effect on productivity of the use of a new tool. You have two groups that are otherwise equal except for use of the tool: group A is using the existing method, while group B is using the tool to perform the designated task. You are measuring productivity in terms of thousands

of delivered source code instructions per month, and you feel confident that the productivity data comes from a normal distribution. You can use Student's t-test to compare group A's productivity data with group B's to see if the use of the tool has made a significant change in productivity.

On the other hand, suppose you want to investigate whether the cleanroom technique yields higher-quality code than your current testing technique. Your hypothesis is stated as

> Code developed using the cleanroom technique has the same number of defects per lines of code as code developed using current testing techniques.

You collect data on the number of defects per thousand lines of code for each of two groups, and you seek an analysis technique that will tell you whether or not the data supports the hypothesis. Here, the data on defects per theousand lines of code is not normally distributed. You can analyze the defect data by ranking it and using the Kruskal–Wallis test to tell if the mean rank of the cleanroom defects is lower than that of the non-cleanroom data.

4.2.2 Exploring a Relationship

Often, an experiment is designed to determine the relationship among data points describing one variable or across multiple variables. For example, you may be interested in knowing the normal ranges of productivity or quality on your projects, so that you have a baseline to compare for the future. A case study may be more appropriate for this objective, but you may want to answer this question as part of a larger experiment. There are three techniques to use to answer questions about a relationship: box plots, scatter diagrams, and correlation analysis.

A *box plot* can depict a summary of the range of a set of data. It shows you where most of the data is clustered and where any outlier data may be. Whereas a box plot shows information about one variable, a *scatter diagram* depicts the relationship between two variables. By viewing the relative positions of pairs of data points, you can visually determine the likelihood of an underlying relationship between the variables. You can also identify data points that are atypical, because they are not organized or clustered in the same way as the other data points.

Correlation analysis goes a step further than a scatter diagram by using statistical methods to confirm whether there is a true relationship between two attributes. Correlation analysis can be done in two ways: by generating measures of association that indicate the closeness of the behavior of the two variables, or by generating an equation that describes that behavior. For example, you may be investigating the relationship between the number of control paths in a module and the number of defects identified in the module. A statistical analysis of these two variables can yield a measure of association between -1 and 1, that indicates whether a high number of control paths usually means a large number of defects. That is, the measure of

association is 1 if there is a perfect positive linear relationship between the two variables, −1 if there is a negative linear relationship, and 0 if there is no relationship. If you want to be able to predict the number of defects from the number of control paths, then the measure of association is sufficient; correlation analysis techniques can generate an equation of the form:

$$(\text{number of defects}) = (\text{constant}_1)(\text{number of control paths}) + (\text{constant}_2)$$

to permit such prediction.

When measures of association are sufficient, it is important to know if the data is normally distributed or not. For normally-distributed values, a *Pearson correlation coefficient* is a measure of association that indicates whether the two variables are highly correlated or not. For non-normal data, you must rank the data and use the *Spearman rank correlation coefficient* as a measure of association. An alternative for non-normal data is the *Kendall robust correlation coefficient*, which investigates the relationship among pairs of data points and can identify partial correlations. If the ranking contains a large number of tied values (that is, repeats of the same value, so that they rank the same), a χ^2 *test on a contingency table* can be used to test the association between the variables.

When you need to understand the nature of the association as well, you can use *linear regression* to generate an equation to describe the relationship between two variables you are examining. This technique produces a line that minimizes the sum of the squares of the residuals. That is, linear regression minimizes the distance from the line to the points off the line. For more than two variables, *multivariate regression* can be used.

The data to which regression is applied does not need to be normally distributed. However, regression techniques assume that the residuals (the distance from each point to the line) are normally distributed. When they are not, two techniques can be used instead of standard regression. *Theil's robust regression* uses the slopes of a carefully-defined set of lines to determine the slope of the regression line. Or, the data points can be transformed to a more normal distribution by viewing their *logarithms*, rather than the data itself; then, regression is applied to the new data to generate a formula, and conversion back to non-logarithmic variables completes the process.

4.2.3 Evaluating the Accuracy of a Model

For many software engineering tasks, a model of behavior is used to predict what should happen. These predictions aid the project manager in making major decisions about resource planning and the duration or extent of activities. For example, the manager may model the likely cost of a project as a basis for decisions about resource allocation. Alternatively, he or she may use a defect model to predict how long to test before turning over the product to the customer. It is important to be able to evaluate the accuracy of these models. In these cases, although the intent of the experiment is different from confirming a theory or exploring a relationship, the

analysis techniques are the same. The model in question generates a set of predicted data, and it is to be compared with the set of actual data. Thus, the steps described in section 4.2.1 can be followed here as well. If the data is normally distributed, a Student's t-test can be used to determine if there is a significant difference between the predictions and the actual numbers. If the data is not normally distributed, then the Kruskal–Wallis test will suffice.

4.2.4 Validating a Measure

Validating a measure means verifying that the measure actually captures the attribute it claims to reflect. For example, the McCabe cyclomatic complexity measure [McCabe 1976] claims to capture the complexity of source code, when in fact it actually measures the number of decision points in the code. Experiments are often designed to validate a measure by exploring the relationship between the measure and data that is known to be highly correlated with the attribute in question. For this reason, the analysis techniques in section 4.2.2 are the appropriate ones for validating a measure.

4.3 Design Considerations

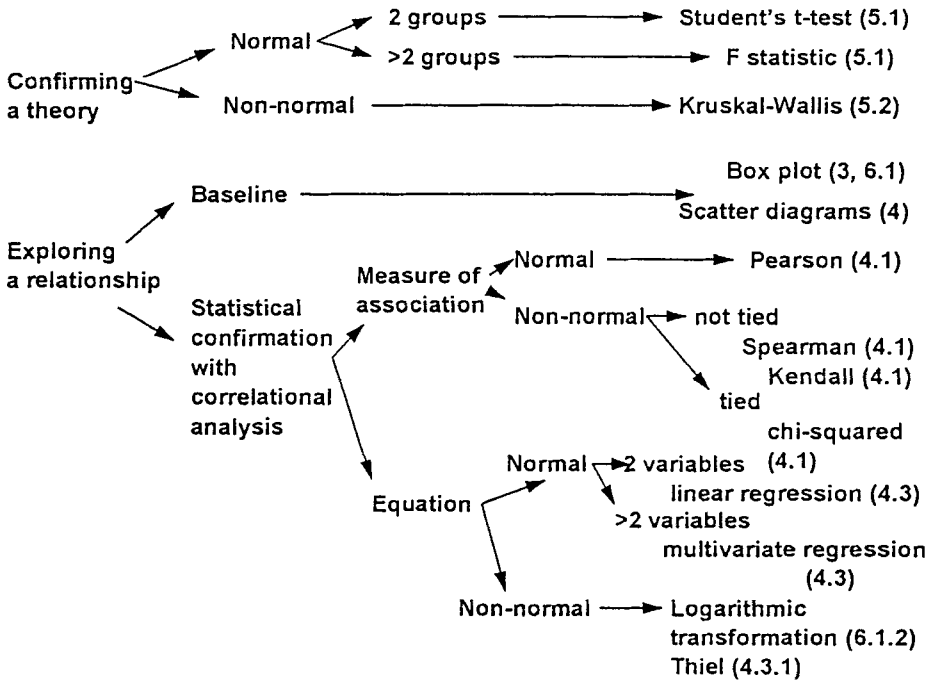
The experimental design must be considered in choosing the analysis techniques. At the same time, the complexity of analysis can influence the design chosen, as noted in section 3. Multiple groups usually generate the need to use the F statistic with a full-blown analysis of variance, rather than a simple Student t-test with two groups. For complex factorial designs with more than two factors, more sophisticated tests of association and significance must be used. Statistical techniques can be used to account for the effects of one set of variables on others, or to compensate for timing or learning effects. These techniques are beyond the scope of this paper, and you should consult a statistician for help with this type of design.

4.4 Decision Tree

To help you understand when to use one analysis technique or another, we have constructed a decision tree to take into account the major considerations discussed in this section (see table 2). The decision tree is to be read from left to right. Beginning with the objective of your experiment, move along the branch that fits your situation until you reach a leaf node with the appropriate analysis technique(s). In parentheses after each analysis technique is a cross-reference to [Kitchenham 1992]; this reference will enable you to read about that technique in more detail, complete with examples. For simplicity, the number of experimental objectives has been reduced from four to two, consistent with the discussion in sections 4.2.3 and 4.2.4.

Table 2

Decision tree for analysis techniques.



5. Summary

We have described the key activities necessary for designing and analyzing an experiment in software engineering. We began by explaining how to choose an appropriate research technique to fit the goals of your project. In particular, we taught you how to state your hypothesis and determine how much control you need over the variables involved. If control is not possible, then a formal experiment is not possible; a case study may be a better approach.

Next, we explained the principles of formal experimentation. We listed the six stages of an experiment: conception, design, preparation, execution, analysis and dissemination. The design of an experiment was discussed in some detail. In particular, we pointed out that you must consider the need for replication, randomization and local control in any experiment that you plan to perform. We showed you how you can think of your design in terms of two types of relationships between factors (crossing and nesting), and we described several issues to be considered when selecting an appropriate design.

Once your experimental design was determined, we discussed how to analyze the results. We explained how the distribution of the data can influence the choice of analysis technique, as can the purpose of the experiment and the design considerations.

This paper should be useful not only in helping you set up your own case studies and experiments, but also in assessing the work of others. There is a profusion of experiments reported in the software engineering literature, many of which suggest that you adopt a particular method, tool or technique. With the analysis suggested here, you should develop a critical eye that will enable you to determine when the reported results are valid and whether the results can be applied to your situation or organization.

References

Caulcutt, R. (1991), *Statistics in Research and Development*, Chapman and Hall, London, England.

Chatfield, C. (1993), *Statistics for Technology*, Chapman and Hall, England.

Fenton, N. S.L. Pfleeger, and R.L. Glass (1994), "Science and Substance: A Challenge to Software Engineers", *IEEE Software 11*, 4, 86–95.

Green, J. and M. d'Oliveira (1990), *Units 16 & 21 Methodology Handbook (part 2)*, Open University, Milton Keynes, England.

Kitchenham, B. (1992), *DESMET Handbook of Data Collection and Metrication Book 3: Analysing Software Data*, National Computer Centre, Manchester, England.

Lee, W. (1975), *Experimental Design and Analysis*, W.H. Freeman and Company, San Francisco, CA.

McCabe, T.J. (1976), "A Complexity Measure", *IEEE Transactions on Software Engineering SE-2*, 4, 308–320.

Ostle, B. and L.C. Malone (1988), *Statistics in Research*, Fourth Edition, Iowa State University Press, Ames, Iowa.

Pfleeger, S.L. (1993), *DESMET Experimental Design and Analysis Procedures (EXPDA)*, National Computing Centre, Manchester, England.