Building Knowledge through Families of Software Studies: An Experience Report[1]

Victor Basili, Forrest Shull, and Filippo Lanubile

Abstract

Experimentation in software engineering is difficult. One reason is that there are a large number of context variables, and so creating a cohesive understanding of experimental results requires a mechanism for motivating studies and integrating results. It requires a community of researchers that can replicate studies, vary context variables, and build abstract models that represent the common observations about the discipline. This paper discusses the experience of the authors, based upon a collection of experiments, in terms of a high level framework for organizing sets of related studies. With such a framework, experiments can be viewed as part of common families of studies, rather than being isolated events. Common families of studies can contribute to higher level hypotheses that no individual experiment could achieve. Then the replication of experiments within a family of studies can act as the cornerstone for building knowledge in an incremental manner. A mechanism is suggested that motivates, records, and integrates individual experiments within a family for analysis by the community at large.

To support the framework, this paper discusses the experiences of the authors in carrying out empirical studies, with specific emphasis on persistent problems encountered in experimental design, threats to validity, criteria for evaluation, and execution of experiments in the domain of software engineering.

1. Introduction

Carrying out empirical work is complex and time consuming; this is especially true for software engineering. Unlike manufacturing, we do not build the same product, over and over, to meet a particular set of specifications. Software is developed and each product is different from the last. So, software artifacts do not provide us with a large set of data points permitting sufficient statistical power for confirming or rejecting a hypothesis. Unlike physics, most of the technologies and theories in software engineering are human based and so variation in human ability tends to obscure experimental effects. Human factors tend to increase the costs of experimentation while making it more difficult to achieve statistical significance. As a result, in software engineering and more generally in computer science, the balance between evaluation of results and development of new theories or technologies is still skewed in favor of unverified proposals [38, 39]. So, empirical research may appear in the software engineering field as a side topic instead of being a standard way to validate claims, as in other scientific disciplines.

Surely empirical software engineering research is not free of flaws. The measurements are not always appropriate to the goals of the experiment, the design does not always avoid alternative explanations of the experimental findings, and the findings are sometimes generalized to a population that is different from the experimental sample [17]. Empirical investigators are challenged to design the best study that the circumstances make possible, trying to rule out all the alternative explanations of the results and generalize those results to the setting of interest. Although the investigators may not get the "perfect" study (assuming there is a perfect one), they have to report the study in such a way that others can verify the conclusions and be aware of the biases that can make the conclusions more equivocal.

The authors have had varying experiences in running a number of empirical studies in software engineering for over twenty years. Most recently, we have worked together to explore and test hypotheses on how software engineers use documents when constructing and analyzing software. While planning, designing, executing, and analyzing our experiments, we have experienced directly how hard it is to run valid studies that provide empirically based conclusions and whose implications are still felt important for software engineers. In this paper, our goal is to (1) discuss the experience of the authors while carrying out empirical

[1] This work was supported by UMIACS and by NSF grant CCR9706151.

studies in the domain of software engineering, and (2) suggest how to package this experience in a form that supports multiple replication studies around common questions of interest.

For the purpose of this paper, we use the definitions of some key terms from [1]. An empirical study, in a broad sense, is an act or operation for the purpose of discovering something unknown or of testing a hypothesis, involving an investigator gathering data and performing analysis to determine what the data mean. This covers various forms of research strategies, including all forms of experiments, qualitative studies, surveys, and archival analyses. An experiment is a form of empirical study where the researcher has control over some of the conditions in which the study takes place and control over the independent variables being studied; an operation carried out under controlled conditions in order to discover an unknown effect or law, to test or establish a hypothesis, or to illustrate a known law. This term thus includes quasi-experiments and pre-experimental designs. A controlled experiment is a specific form of experiment in which the subjects are randomly assigned to experimental conditions, the researcher manipulates independent variables, and the subjects in different experimental conditions are treated similarly with regard to all variables except the independent variables.

One reason experimentation in software engineering is so hard is that the results of almost any process depend to a large degree on a potentially large number of relevant context variables. Because of this, we cannot *a priori* assume that the results of any study apply outside the specific environment in which it was run. Unfortunately, in software engineering, too many studies tend to be isolated and are not replicated, either by the same researchers or by others. For these isolated studies, even if they are themselves well-run, it is difficult to understand how widely applicable the results are, and thus to assess the true contribution to the field.

As an example, consider the following study:

- **Basili/Reiter**. This study was undertaken in 1976 in order to characterize and evaluate the development processes of development teams using a disciplined methodology. The effects of the team methodology were contrasted with control groups made up of development teams using an "ad hoc" development strategy, and with individual developers (also "ad hoc"). High-level hypotheses were proposed: that (BR1) a disciplined approach should reduce the average cost and complexity (faults and rework) of the process and (BR2) the disciplined team should behave more like an individual than a team in terms of the resulting product. The study addressed these high-level hypotheses by evaluating particular methods (such as chief programmer teams, top down design, and reviews) as they were applied in a classroom setting. [7]

This was a rigorous study, but unfortunately never led to a larger body of work on this subject. The specific experiment was not replicated, and the high-level hypotheses were not studied. Thus it was never investigated whether the results hold:

- for software developers at different levels of experience (the original experiment used university students);
- if development teams are composed differently (the original experiment used only 3-person teams);
- if another disciplined methodology had been used (i.e., were the benefits observed due to the particular methodology used in the experiment, or would they be observed for any disciplined methodology?).

The above bullets are examples of "choice points" in the design of the experiment. That is, each of the above bullets represents a specific decision that was made about how the experiment would address the high-level hypotheses. To rigorously test the high-level hypotheses, it would be necessary to run further studies that made different decisions at the choice points. If this can be done systematically, then a body of knowledge can be built up that represents a real understanding of some aspect of software engineering. In this way, high-level hypotheses can serve as a means of organizing a set of related empirical studies.

The careful replication of experiments within such a set of studies is the cornerstone to building knowledge in an incremental manner. An experiment can be repeated as strictly as possible (i.e. using all of the same decisions as the original) just to confirm the previous findings. This increases confidence that the original experiment was correctly run and properly reported, and that the results are repeatable. The design itself can be changed to refute some rival hypothesis or make up for some threat to validity in the earlier study.

The context in which the experiment is run can be changed to investigate the necessary conditions for the occurrence of the findings, trying to answer "in which situations does this technology work?" Replications can be performed by the same researchers to increase their personal confidence in the results or the statistical power of the study. But, most important, replications can be performed by independent investigators with the goal to check and improve the results of previous researchers. Facilitating the spreading of replications of empirical studies is a prerequisite to building a credible body of knowledge in this field.

Of course, the difficulty is that the choice points need to be made explicit, so that researchers can understand how much coverage has already been provided in this area, and what still remains to be done before the high-level hypotheses have been adequately tested. It is necessary to avoid identifying both too many choice points, so that providing a significant amount of coverage becomes neither cost-effective nor interesting, and too few choice points, so that interpretation is still difficult because important variables have not yet been identified. Multiple schemes for identifying and organizing choice points in individual experiments have already been proposed in the literature. For example:

- [10] proposed an organizational framework that consisted of four categories corresponding to phases of experimentation: definition, planning, operation, and interpretation. For each phase, categories of choices were identified which had to be explicitly answered. This framework is most concerned with allowing researchers to define the purpose of the experiment and the object of study. For example, under experimental definition, the researcher was asked to identify the purpose for the study (characterization, evaluation, prediction, motivation), classify the object of study (product, process, model, metric, or theory), and determine the scope of the study (whether single project, multi-project, replicated project, or blocked subject-project). [25] proposed a similar framework but provided different values for the dimensions of the classification, reflecting common concerns for experimenters. For example, the researcher was asked to specify whether the concrete object of study in the experiment was a product technology or a process technology; whether the purpose of the experiment was to evaluate the outcome of a process versus the process itself; and whether the study was focused on a single, specific object of study or on multiple objects.
- [28] presented a framework that placed additional emphasis on experimental variables. For example, under the topic of "subjects" researchers were asked to explicitly report the selection criteria used; the experience, training, and background of the subjects; how ethical issues (such as the right to withdraw from the study) were handled; and how many subjects are required based on the power of the statistical analysis procedure.
- [17] present a list of questions by which empirical studies should be evaluated. These questions in turn suggest a framework for researchers to use in specifying their experiments, since suitable information should be reported to answer each of the questions. The questions concern: whether the research is based on empirical evaluation and data, whether the experiment was designed correctly, whether the study is based on a toy or real situation, whether appropriate measures were used, and whether the study was run for a long enough time.

Our intention here is not to rate or recommend the proposed experimental frameworks. Rather, we note that the important common characteristic of all of these frameworks is that they document the key choices made during experimental design, along with their rationales. This allows other experimenters to understand where different choices could have been made, and raise questions as to the likely outcome of other choices. Because these frameworks provide a mechanism by which different studies can be compared, they help to organize related studies and to tease out the true effects of both the process being studied and the environmental variables.

This mechanism allows the primary question of an experiment to shift from "Is a particular process effective?" to "What are the factors that make a particular process effective or ineffective?" This shift gives us more insight into the field of software engineering as a whole. The real objects of study are no longer (a potentially infinite number of) processes, but (a presumably finite set of) process, product, and environmental factors. Having information at this level paves the way for the "continual improvement" described in [8]: When a particular process is only assessed as being effective or not, it remains a black box. When we identify important factors that relate to its effectiveness, we can use this knowledge as the basis for improving the process (if it is effective) or for creating a new process better suited to the needs of the environment (if the original process turns out to be ineffective).

The above discussion leads us to propose that the following criteria are necessary before we can begin to build up comprehensive bodies of knowledge in areas of software engineering:

1. Sets of high level hypotheses that are of interest to the software engineering community, that provide choice points for the selection of detailed hypotheses;
2. Detailed hypotheses written in a context that allow for a well defined experiment;
3. Context variables, suggested by the hypotheses, that can be changed to allow for variation of the experimental design (to make up for validity threats) and the specifics of the process context;
4. A sufficient amount of information so that the experiment can be replicated and built upon; and
5. A community of researchers that understand experimentation, the need for replication, and are willing to collaborate and replicate.

With respect to the Basili/Reiter study introduced above, we can note that while it satisfied criteria 1, 2 and 4, it failed with respect to criteria 3 and 5. It was not suggested by the authors that other researchers might vary the design or manipulate the processes or criteria used for evaluation although the analysis of the data was varied in a later study [6]). Nor was there a community of researchers willing to analyze the hypotheses even if suggestions for replication had been made.

The high-level hypotheses and the organizational framework selected should be used to choose a focus: i.e., help determine the decisions that should be made at the choice points to define an experiment of interest. The objective is to define a study that contributes to building knowledge in the "big picture," represented by the high-level framework, but in which the specific object of study is still practical and useful for the researcher's own environment. At each choice point, we must be careful to assure ourselves that the problem remains interesting to the software engineering community, i.e., we must pick processes that are of interest and value, and criteria that are meaningful and measurable.

The remainder of this paper is organized according to the five criteria of the organizational framework. We will refer to the studies we have run (as well as others) in order to illustrate what we have learned over time about building knowledge from carefully planned individual studies. As we proceed, we will also use our remarks in this paper to suggest guidelines for lab packages: that is, packages that collect not only experimental artifacts but also experimental experiences for the purpose of encouraging and facilitating carefully planned replications.

2. Formulating High Level Hypotheses and an Experimental Framework

As an example of using high-level hypotheses and an experimental framework to motivate specific, individual studies, we present our work in the field of reading techniques. Reading techniques are procedural techniques, each aimed at a specific development task, which software developers can follow in order to obtain the information they need to accomplish that task effectively [2, 3]. We were interested in studying them in order to determine if beneficial experience and work practices could be distilled into procedural form, and used effectively on real projects.

In order to effectively investigate reading techniques, we needed some way of identifying and organizing important factors that might influence the execution and effectiveness of reading techniques. For example, because reading techniques are each aimed at a specific development task, we wanted some way of identifying important categories of development tasks. We reasoned that this would later allow us to identify attributes of the tasks themselves that effected the reading techniques we created for them. Because reading techniques are also concerned with obtaining the relevant information from a particular source, we wanted to identify pertinent attributes of the software artifacts that would be used in this way.

We developed a tree-structured framework that described the problem space with which we intended to deal [3], and have evolved it as new experiments have been attempted. Figure 1 represents the current version of that framework. The top of the framework represents the problem space and the bottom leaves represent the existing techniques built for empirical study.

At the topmost level, we divided the set of all reading techniques into two groups, depending on whether the task for which they were used was an analysis task (verifying that a software document possessed certain characteristics) or construction task (using a software document to construct a new system). At the next level, we grouped reading techniques according to the specific objective to be achieved within that overall goal, e.g., finding anomalies in the software or making a maintenance modification. Another variable is the document to which the task is applied. Some of these objectives may be applied in different forms to documents at different phases of the lifecycle; e.g., we may undertake anomaly detection in both requirements documents and code, which would require very different techniques. We can also think of some objectives that apply to documents from multiple phases of the lifecycle; e.g., an Object-Oriented framework is an artifact that is advertised to support reuse of both design and code. Finally, the last level of the taxonomy addresses the very specific notation in which the document is written. For example, requirements can be written in a natural language, or translated into a formal notation for requirements such as SCR [21].

The advantage of the tree structure is that more and more detail is specified, as we make more choices and move lower in the tree. This structure allows other researchers to go down the same tree path or branch off at a higher level. This tree structure framework allows us to define classes of experiments and combine them back to high-level hypotheses. Some of these hypotheses are associated with the root of the tree, i.e., with all the techniques at the leaves of the tree. Example root level hypotheses include:

- We can effectively design and study techniques that are procedurally defined, document and notation specific, goal driven, and empirically validated for use.
- We can demonstrate that a procedural approach to a software engineering task could be more effective than a less procedural one.

Other hypotheses can be associated with various branches in the tree. For example, associated with the analysis branch of the tree is the hypothesis that we can create families of related reading techniques such that multiple reviewers can each concentrate on analyzing a specific aspect of the document; the different techniques in the family will together cover all aspects of interest in the entire document. Associated with the construction branch of the tree is the hypothesis that we can create techniques for understanding the document that do not require coverage of the entire software artifact but will be sufficient for the required task goal.

Thus we can define a variety of sub-hypotheses associated with the tree framework. Although each reading technique by necessity is applied to a particular task and document, we can gain confidence in our study of a hypothesis only as we study reading techniques for multiple types of tasks and multiple types of documents. Thus all experiments that fit in the context of this tree can be used to test the hypotheses that procedural techniques for understanding particular documents can be defined and studied and that procedural guidance for reading techniques can be more effective than nonsystematic or ad hoc approaches. Also, each particular experiment can identify specific hypotheses relevant for that experiment, allowing us to build new sets of hypotheses that can be added to the various branches of the framework.

Relating hypotheses through a tree structure helps keep the relevant context variables organized and thus helps in assessing the contribution of a particular experiment. By tracing down the branches of the tree that describe an experiment, the relevant process factors and environmental variables can be easily identified. The tree structure also serves as a basis for comparing experiments, since the location in the tree highlights whether the decisions at the choice points were the same or different for any two experiments. Finally, the tree summarizes the amount of experimentation in an area and illustrates the degree of “coverage” by showing for which branches experiments have already been run, and which still require investigation.

Various techniques have been developed for study and are represented by the roots of the framework in Figure 1. These have been the basis of sets of experiments, representing varying levels of maturity and replication. A subset of the particular experiment set will be used as the basis for discussion in this paper.

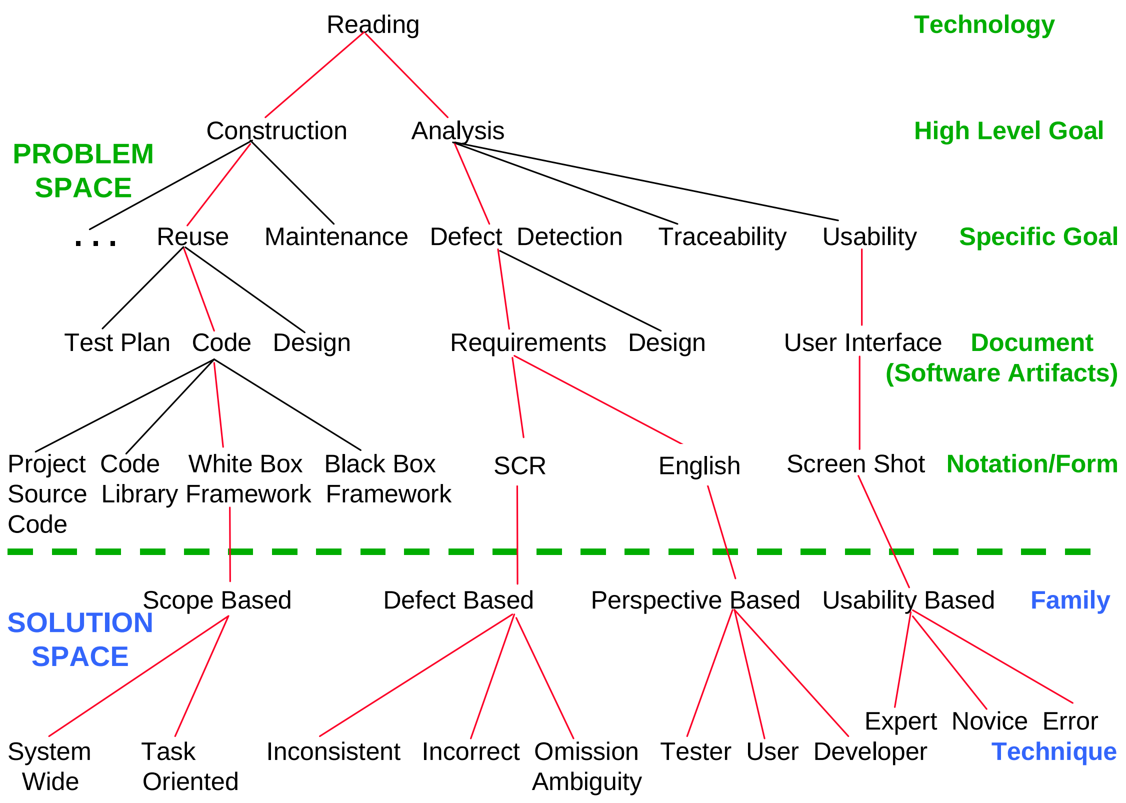

Figure 1: Partial problem space addressed by software reading techniques.

The set of experiments that were suggested by this tree structure include:

- Defect-Based Reading (**DBR**): This was the first family of scenario-based reading to be investigated, and focused on defect detection in requirements, where the requirements were expressed a state machine notation, called SCR. There are three reading techniques in the DBR family, each providing a procedure and set of questions aimed at uncovering a specific class of defects in the SCR notation. In this study, defect-based reading, ad hoc reading and checklist-based reading were evaluated and compared with respect to their effect on fault detection effectiveness in the context of an inspection team. This addressed root-level hypotheses such as whether we can effectively design and study reading techniques and whether a procedural approach to a software engineering task could be more effective than a less procedural one. It also addressed analysis branch hypotheses such as whether we can effectively define a family of techniques, each with a different focus, but whose union provided reasonable coverage of the artifact. A more specific hypothesis was whether DBR represented a more effective way to review SCR-notation requirements than other common techniques. The study was originally run twice in 1993 using graduate students at the University of Maryland [32], and replicated several times, including a study in 1995 using undergraduate students at the University of Bari [19].
- Perspective-Based Reading (**PBR**): This set of experiments also focused on defect detection in requirements, but for requirements expressed in natural language. Again, there are three reading techniques in the PBR family. Each procedure was based upon the perspective of a particular customer of the document, e.g., the user, the designer, the tester. In the experiments run at of NASA Goddard Space Flight Center in 1994 and 1995, perspective based reading was evaluated and compared with the nonsystematic defect detection technique that the subjects already had experience in the organization. Evaluation was with respect to fault detection effectiveness. These studies addressed the same root-level and analysis level hypotheses as DBR but had different specific experimental hypotheses. The study was originally run twice at NASA and has been replicated several times, including the replication conducted by the Fraunhofer Institute for Experimental Software Engineering using professional developers during a pre-project training course. [4, 24]

- Use-Based Reading (**UBR**): This experiment focused on anomaly detection in user interfaces. It is a family of analysis techniques based on three system user perspectives (expert, novice, error handling). In the experiment run at the Bureau of Census, UBR was compared against the usability inspection technique. Specific hypotheses included the effectiveness of the specific techniques to identify certain classes of anomalies. [40]
- Second Version of PBR (**PBR2**): Based on experiences with the previous PBR studies, new versions of the PBR techniques were created. These new techniques were more procedurally oriented versions of the earlier set of PBR techniques. In particular, we made the techniques more specific in all of their steps. They were aimed at testing the root level hypotheses about procedurally defined techniques even more, as well as providing a better opportunity for checking process conformance. In addition, we augmented the technique with additional steps that asked reviewers to reason more deeply about the types of defects they had already found, and whether further defects of that type existed in the requirements. Specific hypotheses had to do with the ability of the techniques to be effective in detecting errors as well as faults and the ability of multiple readers to extract a common set of errors. The new version was evaluated in an experiment at Maryland in 1997. [27]
- Scope-Based Reading (**SBR**): The only experiment so far in testing the construction branch of the tree is the study of scope-based reading. Unlike the analysis branch the hypotheses here are related to finding a particular technique that allows the user to better understand the document for use in constructing another document. Two reading techniques here were developed for learning about an Object-Oriented framework in order to reuse it. The two techniques, one hierarchically based and the other example based, were compared with respect to their ability to support the building of a new system using the framework. Specific hypotheses included whether either of the techniques was more effective for new versus experienced developers. The techniques were observed during use, and hypotheses as to important characteristics of each were built up. [11, 36]

3. Choosing a Specific Focus from the Experimental Framework

As may be noted from the experimental descriptions above, any experiment will investigate very specific hypotheses in addition to the high-level ones. Although the high-level hypotheses and experimental framework may help determine what broad area of the problem domain will be investigated, a specific object of study will have to be selected, a specific context will have to be chosen in which to run the study, and specific metrics will have to be chosen for collection and study.

Choosing the proper specific focus is perhaps the most difficult part of experimentation. Many researchers realize that it can be difficult to experiment when the object of study is not a physical object that can be observed and studied directly, but a relatively nebulous concept such as “process” that can only be studied indirectly, through being executed by the experimental subjects. Because our primary object of study is being observed only indirectly, there are many considerations that can affect the validity of our experiments. Some examples:

- **How do we define/specify the process?**
  This is one of the most critical issues in the study of any process. What is the level of detail in the guidance? Should it vary with the experience of the user of the process? Besides a description of what you should do, should there be a description of what you should not do?

  In an earlier experiment [9], processes were defined for use, e.g., reading by stepwise abstraction, equivalence partitioning boundary value testing, structured testing. Although the procedures may not have been followed directly, the fact that each subject for each technique was given only the particular items needed for the activity, e.g., specification and source code for reading, specification and executables for functional testing, did restrict the subject's ability to perform one process when they were supposed to be performing another.

  It is not difficult to provide a step-by-step process specified in great detail. However, this gives no assurance that this is actually what is done when subjects are asked to execute the process. In [22], the experimenters undertook a study of how people follow processes and determined that users rarely follow even detailed processes step-by-step. Instead, they may “internalize” or follow their own interpretation of the process, and augment the process with information from other sources in order to tailor the process to a given situation. Moreover, we have found in our own experiments that the level of detail at which a process is followed does make a difference.

In the SBR study, subjects were given a process for reusing functionality from example applications. There were characteristic differences between subjects who followed the process strictly and those who used a modified form of it. Both advantages and disadvantages were noted. For example, subjects who followed the process strictly seemed less likely to augment the process when necessary (they were less likely to implement functionality when they could not use the process to find it in an example application, whereas other subjects were able to implement the functionality on their own). Strict followers of the process, however, also seemed less likely to waste time on unnecessary activities (they were less likely to get involved in “gold plating” the new system).

One possible mechanism for helping to get the user to follow the process as defined is to get them to verbalize the process while performing it. One way of doing this might be to have the process performed in pairs, i.e., one person can guide the other.

In a follow up to the original UBR study, we noticed that giving the process to two people to perform as a team, i.e., one reading the rules and recording results, and the other performing the process, may have been more effective in finding anomalies because the process was more rigorously performed.

- **What subjects are performing the process?** In order to have a useful answer to this question, we need to be able to differentiate subjects based on their important characteristics. In order to do this, we have found it useful to employ a mix of qualitative[2] and quantitative methods. Qualitative methods in particular are useful for identifying what the subjects themselves feel may be important experiences or skills that affected their success using the process. The use of methods that elicit subject perspectives are especially important since the experimenters do not always have the correct intuition about what is going on within the subject when a process is applied. The subjects themselves do not always have correct knowledge in this regard, but their intuition and concerns can be a tremendous help in identifying real issues.

  Once potential measures of subject experience have been collected (perhaps via questionnaires or interviews) then quantitative methods can be used to test whether a correlation does exist between any of the experience measures and the effectiveness with which the process is applied. For example:

  While performing PBR, subjects are asked to adopt the point of view of a user of the requirements: either a designer, tester, or user for the system being constructed. Therefore, we expected that the subjects’ previous experience in these roles would be an important factor affecting their effectiveness when using PBR. However, quantitative analysis showed that it was not role experience, but simply the amount of experience with requirements documents, that provided the most important characterization of our subjects. Subjects who had been applying their usual review technique for the longest amount of time found it hard to switch to the new technique, especially for a familiar type of document.

  While important, characterizing subjects is also a difficult task, thanks largely to the large variations seen in human performance [13, 35]. Clear demonstrations of statistically significant relationships between subject characteristics and resulting performance are hard to come by.

- **How do we account for process conformance?** Aside from characterizing the subjects, another feature that needs to be carefully characterized is the execution of the technique itself. It should not be

[2] Qualitative data is information represented as words and pictures, not numbers [20]. Qualitative analysis methods are those designed to analyze qualitative data. Quantitative data, on the other hand, is represented numerically or on some other discrete finite scale (i.e. yes/no or true/false) [34].

assumed that subjects are applying the expected process in the expected way. Subjects are not malicious, but will sometimes concentrate on successfully accomplishing what they see as the goal, even if it means straying from the process assigned. In other cases, the behavior of subjects will be affected by their typical work habits or by techniques with which they have more familiarity, which are not accounted for in the process assigned. Thus experimenters need to worry about process conformance, by placing some bounds on the expected behavior of the subjects applying the process. (This is different from the first bullet which worries about whether the subject can and does perform the process as defined. This is about whether the experimenter knows whether it has been applied, and the level of application). There are multiple approaches for dealing with this: (1) explicitly monitoring the activities of the subjects ourselves, (2) asking subjects to turn in intermediate artifacts they produce as they apply the process, and (3) asking subjects to demonstrate (after the fact) skills they should have gained if they performed the process as expected. Some of these methods have been applied successfully in our previous experiments:

In the SBR study, we built the evaluation of process conformance into the experiment. Because there was not much previous work in the domain of reuse in object-oriented frameworks, we taught the reading techniques to our subjects but did not require their use. Instead, we spent a lot of time and effort monitoring what the subjects did. In this way, we gathered a lot of information about when the techniques were and were not useful, what difficulties were experienced in their use, and what other processes subjects found useful for augmenting the techniques.

In the PBR2 study, we built the evaluation of process conformance into the process. In this experiment, we were more confident in the techniques based on previous experiments. One of our motivations for increasing the level of detail in the techniques was that we could then better assess process conformance. The subjects were required to create intermediate artifacts while following the techniques, which we later collected and studied for insight into how the process was executed. We also asked subjects to cross-reference the defects they found with the specific steps of the technique that had been most useful in their discovery.

Of course, it is important to keep in mind that the process under study and the experiment themselves affect process conformance. For instance, we (as experimenters) might want the subjects to use a very specific process and record many intermediate results, so that we can assess process conformance at a very detailed level. However, if the process is too detailed and requires a high amount of bookkeeping overhead, the process would be tedious and unpleasant for the subjects. The practical usefulness of such a technique should be questioned.

- **How do we select good criteria for effectiveness?** The goal of most studies on process has to do with whether or not the process has a positive effect (the intended effect) on the product. But how do we define the intended effect and how do we measure it? Is the intended effect reduction in cost, improvement in quality? Do we define it directly or by some metric that is easy to measure? A common problem is that we might not fully understand the intended effect or cannot define it sufficiently well to measure it. Therefore we indirectly measure using some metric that we feel confident captures what we mean by effectiveness [18]. A common mistake is to choose a metric that is in reality not as well correlated with our intended effect as we assume. It is necessary therefore to rigorously analyze whether experimental metrics really capture the attributes of interest. An example of this concern occurred in our initial study of PBR:

  Our initial PBR experiment measured the effectiveness of the defect detection processes by means of the number of faults[3] that reviewers discovered. However, is this the correct metric? The ultimate aim

[3] We use the following terms in a very specific way in this paper, based on the IEEE standard terminology [23]. An *error* is a defect in the human thought process made while trying to understand given information, to solve problems, or to use methods and tools. In the context of software requirements specifications, an error is a basic misconception of the actual needs of a user or customer. A *fault* is a concrete manifestation of an error within the software. One error may cause several faults and various errors may cause identical faults. A *failure* is a departure of the operational software system behavior from user expected

of defect detection is to repair the requirements document so as to eliminate misunderstandings of the problem or potential defects in the system . Faults may be too detailed, too specific, too dependent on the reviewer's point of view, and not provide sufficient insight to be well-suited as a basis for repairing requirements. Dealing with a higher level of abstraction (errors) might allow reviewers to identify the really fundamental misconceptions in a document for repair. In the initial analysis of the PBR2 study, subjects reported that errors seemed to convey better information for correcting the document, helped focus reviewers on important areas of the document, and gave a better understanding of the real problems in the requirements.

4. Designing Experiments

In prior sections we have tried to illustrate a framework for designing related families of experiments, and raise relevant issues. Section 2 discussed how a framework relating factors of interest suggests high-level hypotheses. In section 3 we dealt with how these hypotheses and the framework help to determine the particular object and environment of study, and how the object and environment in turn determine the detailed hypotheses that are investigated.

In this section we discuss how unique attributes of experimentation in software engineering determine, sometimes in unexpected ways, the manner in which we are able to explore the hypotheses selected. As examples, we illustrate some of the design decisions we have been faced with in our own experiments, and how these decisions have effected the experimental designs and thus the credibility of conclusions.

4.1. Subject Pool and Experimental Context

A particularly challenging aspect of software engineering research is obtaining subjects for experimentation. Unlike some other fields that study human behavior, empirical software engineering is constrained by the fact that only a relatively small percentage of the human population has the requisite skills to usefully perform software development processes for study and is available for experimentation. For the most representative results, experimental subjects need to be taken from that small percentage. Thus one of the important dimensions describing subjects is that of **experience**. The experience of subjects in skills that are relevant to the object of study must match that of the population to which we want to generalize. However, it is not easy to identify the relevant skills or to measure experience in a way that is meaningful.

In the PBR study, we measured subject experience by asking how many years the reviewer had spent in each of the PBR roles (designer, tester, user). Data analysis revealed that there was a weak relationship between experience and number of defects found, that is, reviewers with more qualification years did not tend to find more defects than reviewers with less or no qualification.

The ideal case of experienced software professionals as experimental subjects is difficult to achieve. Subjects have to be borrowed from a development organization. Because of cost and company constraints, we cannot expect to find as many subjects as we would need to achieve enough statistical power for testing group differences (see [12, 29] for a discussion on statistical power in software engineering experiments).

In the PBR study, in which subjects were software developers from the NASA SEL environment, we estimated the cost per individual would be at least $500 per day and the maximum number of people who might be able to participate in the experiment was 18 subjects. Given these constraints, we minimized the effect of limited experimental subjects by designing a within-subject experiment (subjects were observed multiple times across all the treatments) so that the number of available data points was a multiple of the number of available subjects.

To address the difficulty of obtaining professional subjects, we often use students from software engineering courses. Experiments on students are well-suited to investigating certain issues that do not

---

requirements. A particular failure may be caused by several faults and some faults may never cause a failure. We will use the term *defect* as a generic term, to refer to an error, fault, or failure.

require high levels of industrial experience, e.g. the learning curve associated with training in a new technology. Also, at many universities even undergraduates have relevant industrial experience. Even when the experience level is limited, we can use student experiments to debug experimental protocols before applying the treatment to more expensive, experimental subjects.

Another important dimension is the **experimental context**, especially as it affects subject motivation. In software engineering process studies, we are trying to assess the impact of a technique on real work practices, and we need subjects to perform at a level that is representative of their professional work. Ideally we would like to create an experimental setting for software developers that either reflects their organizational setting or that allows them to see some professional benefit from the activities. This would motivate them to put more effort and thought into activities.

Motivation is a problem when subjects are asked to work on "toy" problems, are given unrealistic processes, or see some other disconnection between the experiment and their professional experience. For our purposes, we group all of these factors under the broad heading of "experimental context". These problems often occur when studies are performed in a "graded classroom setting"[4], where the motivation is course grade, rather than professional need or professional development. The ideal situation would be a training session for a project where the subjects need to learn and build skills in the process for the project they are about to undertake.

Under normal circumstances in classroom experiments, we might expect less process conformance on the part of our subjects, making the results less representative of real development environments. This is especially problematic in cases in which new technologies involve a steep learning curve, since subjects in classroom experiments are typically unlikely to have the motivation to persevere and overcome the learning curve; thus the experiment is unable to measure the benefits of the new technology.

One strategy for improving classroom experiments is to grade the subjects on process conformance rather than results. Although this introduces many problems of its own, we feel that these problems can be identified and overcome in order to yield more representative results. Unfortunately we have not yet found a reliable way to measure process conformance without taking results into account. We have found that subjects, in a graded situation, more likely disregard the experimental protocols if they think it will hinder their chances of being evaluated highly.

In the PBR2 study, we had anticipated investigating whether the type of technique used (PBR or ad hoc) when reviewing requirements affected the number of false positives (i.e. items reported by the reviewer that were not actually defects) that reviewers reported. This was an important question for study, since in "real life" false positives have to be investigated and are then either rejected, or an attempt is made to fix something that wasn't incorrect to begin with. Thus large numbers of false positives imply large amounts of wasted effort. However, it was determined from post-hoc interviews with our subjects that many of them had anticipated that their grade for the assignment would be based on the number of correct defects they found. In order to maximize this number, they reported many questionable defects that ordinarily would not have been included. However, more than anything else this practice increased the number of false positives, and therefore we cannot assume that the count of false positives in this experiment is representative of normal patterns or, indeed, has anything to do with the particular review techniques used.

There are some issues that apply to all types of subjects. For example, even in the ideal case where subjects have high levels of relevant experience and are well-motivated by a professional context, there is the problem of the large degrees of **variability among subjects**. Experienced programmers, even with similar background, greatly vary with respect to their abilities. (We can expect that this is also true for students, if not more so.) Past studies have measured differences in programming performances with high ability subjects who outperform low ability subjects from 4 to 25 times [13]. Considering that in software

[4] The terminology "classroom setting" is troublesome in that it implies that graded classroom experiments and studies on inexperienced student subjects are synonymous. This is not actually the case, as professionals are often students in university courses.

engineering there are no treatments that produce such dramatic effects, high subject differences can easily hide treatment effects with the result of failing to obtain statistically significant effects.

A final issue that affects all types of subjects is that it is very difficult to have a control group. If we have to rely on volunteers, we have to provide some benefit to the subjects and the organization that supports their participation. In cases in which our subjects are students in a class, we have a responsibility to provide some educational benefit to the students as part of their participation. Usually this benefit is provided in terms of training in a new approach. The new approach is the experimental treatment that is being compared to a usual approach, corresponding to the absence of the treatment. Since all subjects must get something out of the participation to the experiment, it is hard to justify having a group that learns nothing new and is asked just to perform as they normally do.

In the PBR experiment, professional software engineers were expecting to learn a new way to review their requirements documents. Therefore it would have been unrealistic to expect to have a control group of subjects who learned nothing new and were used only for comparison purposes.

4.2. Ordering of Events and Activities

Experiments in software engineering are often concerned with assessing effectiveness: How useful some process, notation, or tool will be to software development. Because of a lack of analytical models that describe how people use software processes, assessing an object of study in *absolute* terms is almost never feasible. We lack subjective knowledge as to what are the best metrics to measure, what range of values for these metrics should be considered "good", and what kind of tradeoffs are necessary if we desire to maximize a particular metric, possibly at the expense of others.

The alternative, which is adopted in many software engineering experiments, is to assess effectiveness of a software development technique in comparison to a similar one. Of course, the comparison technique needs to be representative of currently accepted techniques, so that the basis of comparison is well understood.

In an ideal situation, this leads to an experimental design similar to that seen in the Basili/Selby experiment [9], illustrated in Figure 2. Each experimental treatment is performed by each of the subjects. Such a within-subjects design addresses some of the concerns about experimental subjects raised in the preceding section: because each subject effectively serves as his or her own control (i.e. we can measure any improvement for a subject against the baseline of his or her own previous performance) the lack of a control group is mitigated. For the same reason, the variability among subjects' skills is less likely to affect the results. Also, because each subject provides multiple data points we make the best use of the subjects that we do obtain. We can also vary the order in which subjects encounter the treatments so that if there is a learning curve (i.e. subjects get more savvy at applying processes in the experiment regardless of the type of process applied) then later processes do not look more effective than they actually are.

	Group 1	Group 2	Group 3
Code Reading	Pgm 1 / Day 1	Pgm 2 / Day 2	Pgm 3 / Day 3
Functional Testing	Pgm 3 / Day 3	Pgm 1 / Day 1	Pgm 2 / Day 2
Structural Testing	Pgm 2 / Day 2	Pgm 3 / Day 3	Pgm 1 / Day 1

Figure 2: Basili/Selby experimental design

Unfortunately, such a clean design is not always possible. The above design could be used in Basili/Selby because each process examined required certain support artifacts, without which it could not be executed. Thus the experimenters could control when procedures were applied by the availability of support documents. In contrast, let us consider cases in which the learning of one process could actually impact the later execution of other processes, and this interaction cannot be explicitly controlled by the experimenters:

In the PBR study, we wanted to evaluate a systematic technique (PBR) against a nonsystematic one. However, we were afraid that teaching the subjects a new and systematic technique would bias later performance on the nonsystematic comparison technique. That is, we did not feel it possible to give subjects guidance and then ask them to forget it and not use it later, when they were given some freedom to use their own techniques. Especially since the comparison technique was nonsystematic, we considered it likely that some ideas from the new technique might find their way into the application of the comparison technique.

A systematic technique such as PBR, which provides instructions for what actions have to be carried out, might distort the later use of non-systematic techniques in which reviewers are free to find their own way to accomplish the required task. Since it cannot be ruled out that subjects would have continued to use some of the earlier directions even though a different technique was later assigned, the inspection order should conform to an increasing scale of prescriptiveness. That is, teams are not given the chance to apply less prescriptive techniques of their own after learning a more prescriptive technique, such as PBR. This prevents the less prescriptive techniques from incorporating guidance from the more prescriptive ones.

	Group 1	Group 2	
Nonsystematic technique	Doc 1	Doc 2	Day 1
PBR technique	Doc 2	Doc 1	Day 2

Figure 3. The experimental design of the PBR experiment.

One potential problem with the PBR design is that it assumes that subjects do not share knowledge about the documents. (A more complete discussion of this design appears in the next section.) For example, every team in Group 1 reviews document 2 on the second day; if team members receive information about the document from a team who reviewed it on day 1, they would presumably perform more effectively with the technique used on day 2 than they would have normally. This is primarily a concern in classroom experiments; as discussed in section 4.1, the motivation of subjects in this case is somewhat different than that of software professionals, especially when they are being graded on their participation. We address this problem by using unique documents for each treatment in the experiment (see Figure 4 which shows the design of the PBR2 study). In this way, subjects cannot learn any information about a document from each other before encountering it in the experiment, and results are not biased in this way. However, to avoid carryover effects from systematic to nonsystematic techniques, we had all subjects review the document with the ad hoc technique on the first day and with the PBR technique on the second day.

	All subjects	
Ad Hoc technique	Doc 1	Day 1
PBR technique	Doc 2	Day 2

Figure 4. The experimental design of the PBR2 experiment.

4.3. Resulting Threats to Validity

In sections 4.1 and 4.2, we have discussed what we see as the very real constraints imposed by the nature of the software engineering discipline and empirical work. Subjects may not see any benefits for their participation and apply little effort, or may try to maximize their performance in ways not expected by the experimenters. The teaching of new techniques may influence subjects' own ways of doing things.

We have also discussed what we have found to be the most useful and feasible responses on the part of the experimental design to those constraints. We take the view that, at least for the foreseeable future, these are constraints which we as experimentalists will have to live with. Of course, the changes these constraints cause to our experimental designs are not without price. Each of the designs introduced in the last 2 sections has some associated threats to internal validity (i.e. alternative explanations for any differences observed in the process under study). We submit that these threats are not the result of sloppy experimental design, but of constraints unique to the study of human performance in general and software engineering in particular. We have found that our best strategy is to plan related sets of studies which, taken as a whole, can increase confidence that the object of study and not the threats to validity are responsible for any changes observed.

The Basili/Selby design introduced in the last section is one of the few designs with no serious threats to validity but unfortunately is hard to achieve in practice. As we introduce variations on this design we tend to obtain more realistically useful designs at the cost of introducing additional threats. For example, the PBR design (Figure 3) solves the practical problem of combining the execution of systematic and nonsystematic processes in one experiment. However, it introduces a threat since any effect due to time is completely confounded with the effect due to the technique used. We could imagine a *learning effect*, that is reviewers get more adept at finding defects in requirements, no matter what technique is used, and thus do better than normal on the second day, resulting in overestimating the effect due to PBR. Alternately we could imagine a *boredom or tiredness effect*, where reviewers become bored or tired with the experiment over time, and expend less effort on the second day, resulting in underestimating the effect. The point is that these hypotheses, and any number of analogous ones, cannot be dismissed, and exist as potential explanations for any effect seen. (Many of these threats could be avoided, or at least better measured, simply by including a control group that uses PBR both days. However, given the constraints on our subject pool discussed in section 4.1 – namely that volunteer or student subjects expect to gain something of benefit from the experiment in a relatively limited time – we do not accept this as a feasible alternative.)

We argue, however, that the existence of these explanations can be mitigated by taking steps that make them less likely. We can mitigate the existence of a learning curve that increases reviewer effectiveness over time by providing training sessions before the actual treatments of the experiment. In this way, we give subjects the chance to overcome their learning curve during the training sessions rather than the actual experiment.

In the PBR study, the subjects received no feedback regarding their actual defect detection success during the experiment, so that it would presumably be difficult for them to discover whether aspects of their performance were in fact improving their detection rate or not. Furthermore, the documents were dissimilar enough that there little to be learned from the first document that could be transferred to the second.

Similarly, we can overcome some of the other potential effects of time, such as boredom, by scheduling the experiment to give subjects a day off in between the two days of the experiment. This helps prevent subjects from feeling overwhelmed because they must go through the treatments consecutively, and helps avoid the pressures of missing two consecutive days of work.

The design shown in Figure 4 might be viewed as a more difficult case. In this design, not only effects due to time but also effects due to the particular document are confounded with the effect due to the review technique, which is the effect of primary interest. Here, however, there is not so much we can do to mitigate the damage caused by these effects. The primary concern is that documents must be used for which some historical baseline exists so that we have some objective basis for measuring the relative performance of both techniques on the specific document.

A between-subject design that deals with these issues is shown in figure 5. It represents the design in the DBR study. It has the advantage that it allows us to isolate the potential learning effects using a control group but has the problems that one entire group does not get any training in the new process, it requires a larger pool of subjects, and still does not measure the effects of using the systematic process before the non-systematic process.

	Day 1	Day 2
Nonsystematic technique	Doc 1/ Groups 1 & 3	Doc 2/ Group 1
DBR technique	Doc 1/ Group 2	Doc 2/ Groups 2 & 3

Figure 5: Experimental design of DBR with reading technique varying between subjects.

The subjects used in an experiment tend to introduce their own threats, although these are mainly related to *external* validity, i.e. threats concerning how representative the experimental results are of the larger population of software developers. In section 4.2, we have already discussed those threats that come from subjects in classroom experiments, and how we have dealt with them. Studies that use inexperienced reviewers may also face some threats, since it is difficult to say how applicable results on inexperienced subjects are unless the point of the study involves training issues.

We have encountered one final threat to internal validity primarily when working with very experienced reviewers: **interaction of experience and treatment**. In this case, prior experience can interfere with the process that reviewers are asked to execute, so that the process that the experimenters assume is being executed may in fact not be the process that is being applied in reality. For example,

In the PBR experiment, we asked reviewers to apply the PBR technique to documents from both inside and outside their work domain. An improvement due to PBR was seen on the documents from outside the work domain, but not on the ones the subjects were used to working with. The indication from post-hoc interviews with subjects is that this difference may be due to the fact that subjects who had experience reviewing documents in this work domain had built up their own, personal techniques for reviewing such documents. Thus, when experienced reviewers found themselves facing familiar documents the temptation was to revert to their accustomed techniques rather than learning the new technique presented by the experimenters. This tendency was further reinforced by the fact that the experiment had a time limit.

Another form of such interference by preexisting knowledge has been the **spontaneous migration of subjects across treatments**.

In the SBR study, we taught our subjects two different approaches to use a new OO framework so that the strengths and weaknesses of the approaches could be compared. We randomly assigned one half of the class to one approach (Hierarchy-Based), and the remaining half to the other approach (Example-Based). However, we could not prevent the students from using effective work practices that they already knew, or discovered during the project. As a result, we observed that all of the subjects moved away from the Hierarchy-Based approach to some form of Example-Based approach that was found better suited for the problem.

This threat can be mitigated by using checks on process conformance, as discussed in section 3.

5. Replicating Experiments

In preceding sections of this paper, we have tried to raise several reasons why families of replicated experiments are necessary. In this section, we discuss replications in more detail and look at the practical considerations that result. Our primary strategy for supporting replications in practice has been the creation of lab packages, which collect information on an experiment such as the experimental design, the artifacts and processes used in the experiment, the methods used during the experimental analysis, and the motivation behind the key design decisions. Our hope has been that the existence of such packages would simplify the process of replicating an experiment and hence encourage more replications in the discipline. Several replications have been carried out in this manner and have provided us with a growing body of knowledge on reading techniques. We discuss some of these replications in more detail below.

5.1. Types of Replications

Since we consider that replications may be undertaken for various reasons, we have found it useful to enumerate the various reasons, each of which has its own requirements for the lab package. In our view the types of replications that need to be supported can be grouped into 3 major categories:

1. **Replications that do not vary any research hypothesis**
   1.1. **Strict replications** (i.e. replications that duplicate as accurately as possible the original experiment). These replications are necessary to increase confidence in the validity of the experiment. They demonstrate that the results from the original experiment are repeatable, and have been reported accurately by the original experimenters.

   1.2. **Replications that vary the manner in which the experiment is run.** These studies seek to increase our confidence in experimental results by addressing the same problem as previous experiments, but altering the details of the experiment so that certain threats to validity are addressed.

   Due to the relatively small number of subjects and time constraints involved, the PBR study simulated the number of defects that would be found by teams composed of one member using each of the different PBR techniques. A replication was undertaken [15] that varied the experimental design by assigning team members to specific teams and requiring that they meet to agree on a common list of defects. The "simulated" and "real" teams in each of the experiments were used to measure the same thing, but the replication allowed comparison of results between the two methods to provide some confidence in the statistical simulation.

2. **Replications that vary the detailed hypotheses.**
   2.1. **Replications that vary variables intrinsic to the proposed solution.** These replications investigate what aspects of the process are important by systematically varying intrinsic properties of the process and examining the results.

   The version of the requirements review techniques used in the PBR2 study attempted to be as close as possible to the version used in the original experiment, with the exception of the level of detail used. The PBR2 techniques were much more specific, and attempted to constrain the subjects to proven techniques for creating models of the system rather than allowing them to rely on their own techniques. Comparison of the PBR2 results to those from the original experiment (both in terms of defect detection effectiveness and reviewer satisfaction with the technique) will allow us to understand the importance of the level of detail in reading techniques.

   This type of experiment requires the process and artifacts to be supplied in sufficient detail that changes can be made. This implies that the original experimenters must provide the rationales for the design decisions made as well as the finished product.

   2.2. **Replications that vary variables in the environment in which the solution is evaluated.** These studies can identify potentially important environmental factors that can affect the results of the process under investigation.

   The DBR studies have been independently replicated, so far, in five different contexts. Table 1 *summarizes the environmental variables and reports whether the study findings supported the research hypothesis that DBR is more effective than ad hoc reading techniques. Some replications used software practitioners, while others used undergraduates or graduate students, and obtained different results. These studies, when put together, allow us to hypothesize that user experience plays a role. Indications are that DBR is too sophisticated to be successfully applied by undergraduate students (DBR requires the ability to model some aspects of the system being reviewed). The one replication using practitioners allows us to hypothesize that practitioners, as previously discussed in section 4.3, may revert back to the techniques with which they are more familiar. These conditions appear not to hold with graduate students, because they satisfy the skill prerequisites for the reading technique and are not biased by daily working practices.*

3. **Replications that vary the high-level hypotheses.** These replications help determine the limits to the effectiveness of a process, by manipulating the process and/or experimental context variables to see if basic principles still hold.

   The PBR and DBR studies both found somewhat similar results, in that in both cases a family of prescriptive analysis techniques, each based on a particular focus, was found to be more effective than less prescriptive techniques at finding defects in requirements. The similarity in results gave us more confidence as to the effectiveness of focused reading techniques for finding defects in requirements, and showed that the positive effects of such reading techniques were not limited to requirements in a formal notation (DBR) but could be applied to natural language documents (PBR) as well.

   The SBR study is another example. We had also created a family of reading techniques for this problem domain (reuse of functionality from an OO framework) but found that, in this case, the techniques did not complement each other as in the PBR and DBR families. This allowed us to hypothesize about some differences between techniques for analysis tasks and techniques for construction tasks.

Site	No. of Runs	Subjects	Results	Reference:
University of Maryland	2	Graduate students	Positive evidence	[32]
Lucent Technologies	1	Practitioners	Positive evidence	[31]
University of Bari	1	Undergraduate students	No evidence	[19]
University of Strathclyde	2	Undergraduate students	No evidence	[30]
University of Linkoeping	1	Undergraduate students	No evidence	[33]

Table 1. Status of replications of DBR studies

5.2. Implications for Lab Package Design

To date, much of the discussion of reuse between experiments has been focused on the reuse of physical artifacts and concrete processes. This is indeed a useful beginning. The cost of an experiment is greatly increased if the preparation of multiple artifacts is necessary. Creating artifacts which are representative of those used in real development projects is difficult and time consuming. Reusing artifacts can thus reduce the time and cost needed for experimentation. A more significant benefit is that reuse allows the opportunity to build up knowledge about the actual use of particular, non-trivial artifacts in practice. Thus replications (and experimentation in general) could be facilitated if there were repositories of reusable artifacts of different types (e.g. requirements) which have a history of reuse and which, therefore, are well understood. (A model for such repositories could be the repository of system architectures [16], where the relevant attributes of each design in the repository are known and described.)

A first step towards this goal is the construction of web-based laboratory packages. At the most basic level, these packages allow an independent experimenter to download experimental materials, either for reuse or for better understanding. In this way, these packages support strict replications (as defined in section 5.1), which require that the processes and artifacts used in the original experiment be made available to independent researchers.

However, web-based lab packages should be designed to support more sophisticated types of replications as well. For example, packages should assist other experimenters in understanding and addressing the threats to validity in order to support replications that vary some aspects of the experimental setup. Many of the examples in this paper have tried to demonstrate that, due to the constraints imposed by the setting in which software engineering research is conducted, it is almost never possible to rule out every single threat to validity. Choosing the "least bad" set of threats given the goal of the experiment is necessary. Lab packages need to acknowledge this fact and make the analysis of the constraints and the threats to validity explicit, so that other studies may use different experimental designs (that may have other threats to validity of their own) to rule out these threats.

Replications that seek to vary the detailed hypotheses have additional requirements if the lab package is to support them as well. For example, in order for other experimenters to effectively vary attributes of the object of study, the original process must be explained in sufficient detail that other researchers can draw their own conclusions about key variables. Since it is unreasonable to expect the original experimenters to determine all of the key variables *a priori*, lab packages must provide rationales for key experimental context decisions so that other experimentalists can determine feasible points of variation of interest to themselves. Similarly, lab packages must specify context variables in sufficient detail that feasible changes to the environment can be identified and hypotheses made about their effects on the results.

Finally, for supporting the investigation of high-level hypotheses, researchers should know which experiments have been run that offer results related to the hypotheses. Therefore, lab packages for related experiments should be linked, in order to collect different experiments that address different areas of the problem space, and contribute to answering higher-level questions. The web is an ideal medium for such packages since links can be added dynamically, pointing to new, related lab packages as they become available. Thus it is to be hoped that lab packages are "living documents" that are changed and updated to reflect our current understanding of the experiments they describe.

Lab packages have been our preferred method for feeding results and experiences from well-designed studies back to the high-level framework (discussed in section 2). Interested readers are referred to existing examples of lab packages: [5, 28].By collecting detailed information and results on specific experiments, they summarize our knowledge about specific processes. They record the design and analysis methods used and may suggest new ones. Additionally, by linking related studies they can help experimenters understand what factors do or do not impact effectiveness.

5.3. The Experimental Community

A group of researchers, from both industry and academia, has been organized since 1993 for the purpose of facilitating the replication of experiments. The group is called ISERN, the International Software Engineering Research Network, and includes members in North America, Europe, Asia, and Australia . ISERN members publish common technical reports, exchange visitors, and organize annual meetings to share experiences on software engineering experimentation[5]. They have begun replicating experiments to better understanding the success factors of inspection and reading.

[5] More information is available at the URL http://wwwagse.informatik.uni-kl.de/ISERN/isern.html

The *Empirical Software Engineering* journal has also helped build an experimental community by providing a forum for publishing descriptions of empirical studies and their replications. An especially noteworthy aspect of the journal is that it is open to publishing replicated studies that, while rigorously planned and analyzed, yield unexpected results that did not confirm the original study. Although it has traditionally been difficult to publish such "unsuccessful" studies in the software engineering literature, this knowledge must be made available if the community is to build a complete and unbiased body of knowledge concerning software technologies.

Finally we must note that this community has undertaken families of replications which have been very successful at this kind of knowledge-building. One example is the family of DBR studies summarized in Table 1, which have investigated the DBR techniques in a variety of contexts and with a variety of types of subjects. A second example is the series of empirical studies into PBR.

The original study at the University of Maryland has been replicated at the University of Kaiserslautern, Germany, in a study that used a different design to directly study the effects of PBR on reviewer teams [15]. A second replication was performed at the University of Trondheim, Norway, which used a very similar design but altered the PBR techniques in order to study process conformance issues [37]. Although this experiment did not see the expected effects, the ideas raised were very influential in a redesign of the PBR techniques, again at Maryland, in order to address process conformance. This version of the techniques was the basis for the PBR2 experiment, which has been, or is being, replicated at the University of Bari, Italy; Drexel University, U.S.A.; Universidade de Sao Paulo, Brazil; and Lund University, Sweden.

6. Conclusions

It is our contention that high-level hypotheses can be investigated effectively if empirical work is organized in the form of experiment families. In this paper, we have raised several reasons why families of related experiments are necessary:

- To investigate the effects of alternative decisions at different high-level "choice points" (section 2);
- To vary the strategy with which detailed hypotheses are investigated (section 3);
- To make up for certain threats to validity that often arise in realistically designed experiments (section 4).

The set of experiments on reading, discussed in a working group at the 1997 annual meeting of ISERN [26], is an example that we have built up a body of knowledge by independent researchers working on different parts of the problem and exposing their conclusions to different plausible rival hypotheses. We have shown in this paper that experimental constraints in software engineering research make very difficult, and even impossible, to design a perfect single study. In order to rule out the threats to validity, it is more realistic to rely on the "parsimony" concept rather than being frustrated because of trying to completely remove them. This appeal to parsimony is based on the assumption that the evidence for an experimental effect is more credible if that effect can be observed in numerous and independent experiments each with different threats to validity [14].

A second conclusion is that empirical research must be a collaborative activity because of the huge number of problems, variables, and issues to consider. This complexity can be faced with extensive brainstorming, carefully designing complementary studies that provide coverage of the problem and solution space, and reciprocal verification.

Discussion within the experimental community is also needed to address other issues, such as what constitutes an "acceptable" level of confidence in the hypotheses that we address as a community. By running carefully designed replications, we can address threats to validity in specific experiments and accumulate evidence about high-level hypotheses. However, we are unaware of any useful and specific guidelines that concern the amount of evidence that must be accumulated before conclusions can confidently be drawn from a set of related experiments, in spite of the existence of specific threats. More discussion within the empirical software engineering community as to what constitutes a sufficient body of credible knowledge would be of benefit.

Acknowledgements

The authors would like to thank Michael Fredericks and Shari Lawrence Pfleeger for their valuable comments on this paper.

References

[1] V. R. Basili, "The role of experimentation in software engineering: past, current and future", in *Proc. of the 18th Int. Conf. on Software Engineering*, Berlin, Germany, pp. 442-449, 1996.

[2] V. R. Basili, "Evolving and packaging reading technologies", *Journal of Systems and Software*, vol. 38, no. 1, pp.3-12, July 1997.

[3] V. Basili, G. Caldiera, F. Lanubile, and F. Shull, "Studies on reading techniques", *Proc. of the Twenty-First Annual Software Engineering Workshop*, SEL-96-002, Goddard Space Flight Center, Greenbelt, Maryland, December 1996, pp.59-65

[4] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Soerumgaard, M. Zelkowitz, “The empirical investigation of perspective-based reading”; *Empirical Software Engineering Journal*, vol. 1, no. 2, 1996.

[5] V. R. Basili, S. Green , O. Laitenburger, F. Lanubile, F. Shull, S. Sørumgård, and M. Zelkowitz, "Packaging researcher experience to assist replication of experiments", *Proc. of the ISERN meeting 1996*, Sydney, Australia, 1996.

[6] V. R. Basili, and D. H. Hutchens, "An empirical study of a syntactic metric family", *IEEE Transactions on Software Engineering*, vol. SE-9, pp.664-672, November 1983.

[7] V. R. Basili, and R. W. Reiter, "A controlled experiment quantitatively comparing software development approaches", *IEEE Transactions on Software Engineering*, vol. SE-7, no. 3, pp.299-320, May 1981.

[8] V. R. Basili, and H. D. Rombach, "The TAME project: Towards improvement-oriented software environments", *IEEE Transactions on Software Engineering*, vol. SE-14, no. 6, June 1988.

[9] V. R. Basili, R. Selby, “Comparing the effectiveness of software testing strategies”, *IEEE Transactions on Software Engineering*, vol. SE-13, no. 12, pp.1278-1296, December 1987.

[10] V. R. Basili, R. W. Selby, and D. H. Hutchens, “Experimentation in software engineering”, *IEEE Transactions on Software Engineering*, vol. SE-12, no. 7, pp. 733-743, July 1986.

[11] V. Basili, F. Lanubile, F. Shull, "Investigating maintenance processes in a framework-based environment", *Proc. of the Int. Conf. on Software Maintenance*, Bethesda, Maryland, 1998 (to appear).

[12] L. Briand, K. El Emam, and S. Morasca, "On the application of measurement theory in software engineering", *Empirical Software Engineering Journal*, vol. 1, no. 1, pp.61-88, 1996.

[13] R. Brooks, "Studying programmer behavior experimentally: The problems of proper methodology", *Communications of the ACM*, vol. 23, no. 4, pp. 207-213, 1980.

[14] D. T. Campbell, and J. C. Stanley, *Experimental and Quasi-Experimental Designs for Research*, Boston: Houghton Mifflin Co, 1963.

[15] M. Ciolkowski, C. Differding, O. Laitenberger, and J. Munch, "Empirical investigation of perspective-based reading: a replicated experiment", *Technical Report ISERN-97-13*, International Software Engineering Research Network, 1997.

[16] Composable Systems Group, "Model Problems", http://www.cs.cmu.edu/~Compose/html/ModProb/, 1995.

[17] N. Fenton, S. L. Pfleeger, and R. Glass, "Science and substance: A challenge to software engineers", *IEEE Software*, pp. 86-95, July 1994.

[18] N. Fenton, and S. L. Pfleeger, *Software Metrics; A Rigorous and Practical Approach, 2nd edition*, International Thomson Computer Press, London, UK, 1997.

[19] P. Fusaro, F. Lanubile, and G. Visaggio, "A replicated experiment to assess requirements inspections techniques", *Empirical Software Engineering Journal*, vol.2, no.1, pp.39-57, 1997.

[20] J. Gilgun. “Definitions, Methodologies, and Methods in Qualitative Family Research”. In J. Gilgun, K. Daly, G. Handel, editors, Qualitative Methods in Family Research. Sage Publications, 1992.

[21] K. L. Heninger, "Specifying software requirements for complex systems: new techniques and their application", *IEEE Transactions on Software Engineering*, vol. SE-6, no. 1, pp. 2-13, 1980.

[22] G. Hidding, “Reinventing Methodology”, *Communications of the ACM*, vol. 40, no. 11, pp. 102-109, 1997.

[23] IEEE. Software Engineering Standards. IEEE Computer Society Press, 1987.

[24] O. Laitenberger, and J. M. DeBaud, "Perspective-based reading of code documents at Robert Bosch GmbH", *Journal of Information and Software Technology*, 39, pp.781-791, 1997.

[25] F. Lanubile, "Empirical evaluation of software maintenance technologies", *Empirical Software Engineering Journal*, vol.2, no.2, pp.95-106, 1997.

[26] F. Lanubile, "Report on the results of the parallel project meeting reading techniques", http://seldi2.uniba.it:1025/isern97/readwg/index.htm , October 1997.

[27] F. Lanubile, F. Shull, V. Basili, "Experimenting with error abstraction in requirements documents", *Proc. of the 5th Int. Symposium on Software Metrics*, Bethesda, Maryland, 1998 (to appear).

[28] C. M. Lott, and H. D. Rombach, "Repeatable software engineering experiments for comparing defect-detection techniques", *Empirical Software Engineering Journal*, vol.1, no.3, pp.241-277, 1996.

[29] J. Miller, J. Daly, M. Wood, M. Roper, and A. Brooks, "Statistical power and its subcomponents - missing and misunderstood concepts in software engineering empirical research", *Journal of Information and Software Technology*, 39, pp.285-295, 1997.

[30] J. Miller, M. Wood, and M. Roper, "Further experiences with scenarios and checklists", *Empirical Software Engineering Journal*, vol.3, no.1, pp.37-64, 1998.

[31] A. Porter, and L. Votta, " Comparing detection methods for software requirements inspections: a replicated experiment using professional subjects", *Empirical Software Engineering Journal*, 1997 (under review).

[32] A. Porter, L. Votta, V. Basili, “Comparing detection methods for software requirements inspections: a replicated experiment”, *IEEE Transactions on Software Engineering*, vol. 21, no. 6, pp. 563-575, 1995.

[33] K. Sandahl, J. Karlsson, K. Krysander, M. Lindvall, and N. Ohlsson, "A cross-cultural replication of an experiment for assessing methods for software requirements inspections", *Empirical Software Engineering Journal*, 1997 (under review).

[34] Communication and Organization: An Empirical Study of Discussion in Inspection Meetings. Carolyn B. Seaman and Victor R. Basili, IEEE Transactions on Software Engineering, 24(6), June 1998.

[35] B. A. Sheil, “The psychological study of programming”, *ACM Computing Surveys*, 13:1, 101-120, 1981.

[36] F. Shull, F. Lanubile, and V. R. Basili, "Investigating Reading Techniques for Framework Learning", *Technical Report CS-TR-3896*, UMCP Dept. of Computer Science, *UMIACS-TR-98-26*, UMCP Institute for Advanced Computer Studies, *ISERN-98-16*, International Software Engineering Research Network, May 1998.

[37] S. Sørumgård, "An empirical study of process conformance", *Proc. of the Twenty-First Annual Software Engineering Workshop*, SEL-96-002, Goddard Space Flight Center, Greenbelt, Maryland, December 1996, pp.115-124.

[38] W. F. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz, “Experimental evaluation in computer science: a quantitative study”, *The Journal of Systems and Software*, 28, pp. 9-18, 1995.

[39] M. V. Zelkowitz, and D. R. Wallace, "Experimental models for validating technology", *Computer*, pp. 23-31, May 1998.

[40] Z. Zhang, V. Basili, and B. Shneiderman, “An Empirical Study of Perspective-based Usability Inspection”, accepted at Human Factors and Ergonomics Society Conference ’98.