Using Experiments to Build a Body of Knowledge

Victor Basili
Fraunhofer Center Maryland
and Computer Science Dept.
University of Maryland
College Park, MD 20742, USA
basili@cs.umd.edu

Forrest Shull
Institute for Advanced Computer Studies
Computer Science Dept.
University of Maryland
College Park, MD 20742, USA
fshull@cs.umd.edu

Filippo Lanubile
Dipartimento di Informatica
Universita' di Bari
Via Orabona, 4
70126 Bari, Italia
lanubile@di.uniba.it

Abstract

Experimentation in software engineering is important but difficult. One reason it is so difficult 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. This paper argues for the necessity of a 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 important and relevant hypotheses that may not be suggested by individual experiments. A framework also facilitates building knowledge in an incremental manner through the replication of experiments within families of studies.

Building knowledge in this way 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 also presents guidelines for lab packages, meant to encourage and support replications, that encapsulate materials, methods, and experiences concerning software engineering experiments.

1. Introduction

Experimentation in software engineering is necessary. Common wisdom, intuition, speculation and proofs of concepts are not reliable sources of credible knowledge. On the contrary, progress in any discipline involves building models that can be tested, through empirical study, to check whether the current understanding of the field is correct[1]. Progress comes when what is actually true can be separated from what is only believed to be true. To accomplish this, the scientific method supports the building of knowledge through an iterative process of model building, prediction, observation, and analysis. It requires that no confidence be placed in a theory that has not stood up to rigorous deductive testing [21]. That is, any scientific theory must be (1) falsifiable, (2) logically consistent, (3) at least as predictive as other competing theories, and (4) its predictions have been confirmed by observations during tests for falsification. According to Popper, a theory can only be shown to be false or not yet false; researchers only become confident in a theory when it has survived numerous attempts made at its falsification. This paradigm is a necessary step for ensuring that opinion or desire does not influence knowledge.

Experimentation in software engineering is difficult. 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

[1] For the purpose of this paper, we use the definitions of some key terms from [15] and [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 test a hypothesis against observation. This term thus includes quasi-experiments and pre-experimental designs.
A *theory* is a possible explanation of some phenomenon. Any theory is made up of a set of hypotheses. A *hypothesis* is an educated guess that there exists (1) a (causal) relation among constructs of theoretical interest; (2) a relation between a construct and observable indicators (how the construct can be observed).
A *model* is a simplified representation of a system or phenomenon; it may or may not be mathematical or even formal; it can be a theory.

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.

Abstracting conclusions from empirical studies in software engineering research is difficult. An important reason why 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. For 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"). 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 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 study, like any other, required the experimenters to construct models of the processes studied, models of effectiveness, and models of the context in which the study was run. Replications that alter key attributes of these models are then necessary to build up knowledge about whether the results hold under other conditions. Unfortunately, in software engineering, too many studies tend to be isolated and are not replicated, either by the same researchers or by others. Basili/Reiter 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 applicability of the hypotheses in other contexts was not studied. Thus it was never investigated whether the results hold, for example:

- 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?).

2. A Motivating Example: Software Reading Techniques

Yet even when replications *are* run, it's hard to know how to abstract important knowledge without a framework for relating the studies. To illustrate, we present our work on 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 reading techniques in order to determine if beneficial experience and work practices could be distilled into procedural form, and used effectively on real projects. We felt that reading techniques were of relevance and value to the software engineering community, since reading software documents (such as requirements, design, code, etc.) is a key technical activity. Developers are often called upon to read software documents in order to extract specific information for important software tasks, e.g. to read a requirements document in order to find defects during an inspection, or an Object-Oriented design in order to identify reusable components. However, while developers are usually taught how to *write* software documents, the skills required for effecting *reading* are rarely taught and must be built up through experience. In fact, we felt that research into reading could provide a model for how to effectively write documents as well: by understanding how readers perform more effectively it may be possible to write documents in a way that facilitates the task.

However, the concept of reading techniques cannot be studied in isolation. Like any other software process, reading techniques must be tailored to the environment in which they are run. Our aim in this research was to generate sets of reading techniques that were procedurally defined, tailorable to the environment, aimed at accomplishing a particular task, and specific to the particular document and notation on which they would be applied. This has led a series of studies in which we evaluated the following types of reading techniques:

- Defect-Based Reading (**DBR**) focused on defect detection in requirements, where the requirements were expressed using a state machine notation called SCR [13, 22].
- Perspective-Based Reading (**PBR**) also focused on defect detection in requirements, but for requirements expressed in natural language [4, 16].
- Use-Based Reading (**UBR**) focused on anomaly detection in user interfaces [27].
- Second Version of PBR (**PBR2**) consisted of new techniques that 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 [24].
- Scope-Based Reading (**SBR**) consisted of two reading techniques that were developed for learning about an Object-Oriented framework in order to reuse it [10, 23].

A framework that makes explicit the different models used in these experiments would have many benefits. Such a framework would document the key choices made during experimental design, along with their rationales. The framework could be used to choose a focus for future studies: i.e., help determine the important attributes of the models used in an experiment, and which should be held constant and which varied in future studies. The ultimate objective is to build up a unifying theory by creating a list of the specific hypotheses investigated in an area, and how similar or different they all are.

Using an organizational framework also allows other experimenters to understand where different choices could have been made in defining models and hypotheses, and raises questions as to their likely outcome. 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.

3. The GQM Goal Template as a Tool for Experimentation

Examples of such organizational frameworks do exist in the literature, e.g. [9, 17, 20]. For the purpose of this paper we find the Goal/Question/Metric (GQM) Goal Template [8] useful. The GQM method was defined as a mechanism for defining and interpreting a set of operational goals using measurement. It represents a top-down systematic approach for tailoring and integrating goals with models of software processes, products, and quality perspectives, based upon the specific needs of a project and organization.

The GQM goal template is a tool that can be used to articulate the purpose of any study. It ties together the important models, and provides a basis against which the appropriateness of a study's specific hypotheses, and dependent and independent variables, may be evaluated. There are five parameters in a GQM goal template:

- *object of study*: a process, product or any other experience model
- *purpose*: to characterize (what is it?), evaluate (is it good?), predict (can I estimate something in the future?), control (can I manipulate events?), improve (can I improve events?)
- *focus*: model aimed at viewing the aspect of the object of study that is of interest, e.g., reliability of the product, defect detection/prevention capability of the process, accuracy of the cost model
- *point of view*: e.g., the perspective of the person needing the information, e.g., in theory testing the point of view is usually the researcher trying to gain some knowledge
- *context*: models aimed at describing environment in which the measurement is taken

For example, the goal of the Basili/Reiter study, previously described, might be instantiated as:

To analyze the *development processes of a 1) disciplined-methodology team approach, 2) ad hoc team approach, and 3) ad hoc individual approach*
for the purpose of *characterization and evaluation*

with respect to *cost and complexity (faults and rework) of the process*
from the point of view of the *developer and project manager*
in the context of *an advanced university classroom*

Due to the nature of software engineering research, instantiated goals tend to show certain similarities. The *purpose* of studies is often evaluation; that is, researchers tend to study software technologies in order to assess their effect on development. For our purposes, the *point of view* can be considered to be that of the researcher or knowledge-builder. While studies can be run from the point of view of the project manager, i.e. requiring some immediate feedback as to effects on effort and schedule, published studies have usually undergone additional, post-hoc analysis.

The remaining fields in the template require the construction of more complicated models, but still show some similarities. The *object of study* is often (but not always) a process; researchers are often concerned with evaluating whether or not a particular development process represents an improvement to the way software is built. (E.g.: Does Object-Oriented Analysis lead to an improved implementation? Does an investment in reviews lead to less buggy, more reliable systems? Does reuse allow quality systems to be built more cheaply?) When the object of study is a process, the *focus* of the evaluation is the process' effect. The experimenter may measure its effect on a product, that is, whether the process leads to some desired attribute in a software work product. Or, the experimenter may attempt to capture its effect on people, e.g. whether practitioners were comfortable executing the process or found it tedious and infeasible. Finally, the *context* field should include a large number of environmental variables and therefore tends to exhibit the most variability. Studies may be run on students or experts; under time constraints, or not; in well-understood application domains, or in cutting-edge areas. There are numerous such variables that may influence the results of applying a technique.

For the remainder of this paper, we will illustrate our conclusions by concentrating on studies that investigate process characteristics with respect to their effects on products. A GQM template for this class of studies is:

> Analyze processes to evaluate their effectiveness on a product from the point of view of the knowledge builder in the context of (a particular variable set).

For particular studies in this class, constructing a complete GQM template requires making explicit the process (object of study), the effect on the product (focus), and context models in the experiment. Making these models explicit is necessary in order to understand the conditions under which the experimental results hold.

For example, consider the GQM templates for the list of reading technique experiments described in the previous section. There are many ways of classifying processes, but we might first classify processes by scope as:

- Techniques (processes that can be followed to accomplish some specific task),
- Methods[2] (processes augmented with information concerning when and how the process should be applied),
- Life Cycle Models (processes which describe the entire software development process).

Each of these categories could be subdivided in turn. The set of techniques, for example, could be classified based on the specific task as: Reading, Testing, Designing, and so on. We have found it helpful to think of the range of values as organized in a hierarchical fashion, in which more general values are found at the top of the tree, and each level of the tree represents a new level of detail. (Figure 1)

Selecting a particular type of process for study, our GQM template then becomes:

> Analyze reading techniques to evaluate their effectiveness on a product from the point of view of the knowledge builder in the context of a particular variable set

[2] The definitions of "technique" and "method" are adapted from [5].

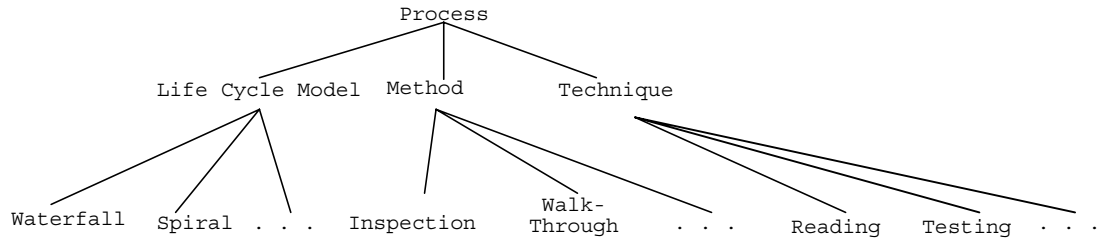

Figure 1: A portion of the hierarchy of possible values for describing software processes.

The reading technique experiments were concerned with studying the effect of the reading technique on a product. So, the model of focus needs to specify both how effectiveness is to be measured and the product on which the evaluation is performed.We find it useful to divide the set of effectiveness measures into analysis and construction measures, based on whether the goal of the process is to analyze intrinsic properties of a document or to use it in building a new system. Each of these categories can be further broken down into more specific types of process goals, for which different effectiveness measures may apply (Fig. 2). For example, the effectiveness of a process for performing maintenance can be evaluated by how that process effects the cost of making a change to the system. The effectiveness of a process for detecting defects in a document can be measured by the number of faults it helps find. Of course, many more measures exist than will fit into Figure 2. For instance, rather than measure the number of faults a defect detection process yields, it might be more appropriate to measure the number of errors[3], or the amount of effort required, among other things.

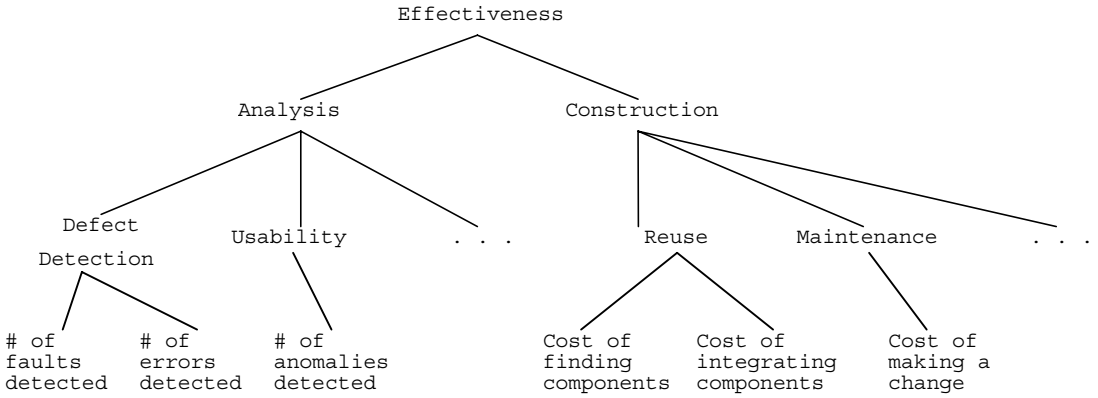

Figure 2: A portion of the hierarchy of possible values for describing the effectiveness of software processes

Similarly, a software document can be classified according to the model of a software system it contains (a relatively well-defined set) and further subdivided into the specific notations that may be used (Fig.3). The main purpose of organizing the possible values hierarchically is to organize a conception of the problem space that can be used by others for classifying their own experiments. The actual criteria used are somewhat subjective; naturally there are multiple criteria for classifying processes, effectiveness measures, and software documents, but we have selected just those that have contributed to our conception of reading techniques.

[3] Here we are using the terms "faults" and "errors" according to the IEEE standard definitions [14], in which "fault" refers to defects appearing in some artifact while "error" refers to an underlying human misconception that may be translated into faults.

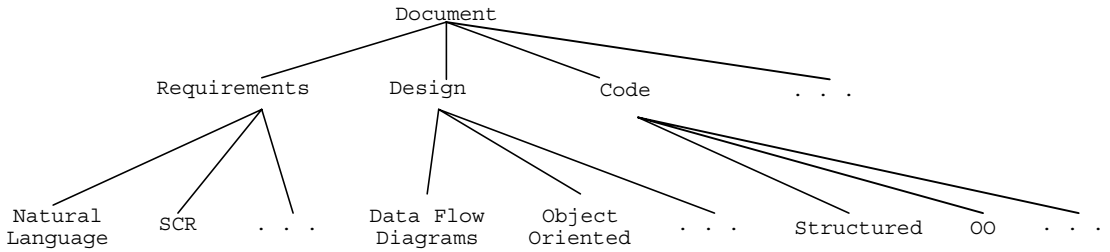

Figure 3: A portion of the hierarchy of possible values for describing software documents.

Thus a GQM template for the PBR experiment could be:

> Analyze reading techniques to evaluate their ability to detect defects in a Requirements Document written in English from the point of view of the knowledge builder in the context of a particular variable set .

A GQM goal is not meant to be a definitive description, but reflects the interests and priorities of the experimenter. If we were to study the process model for the reading techniques in each experiment in more detail, we would see that each technique is tailored to a specific task (e.g., analysis or construction, etc.) and to a specific document. This is what characterizes the reading techniques and distinguishes them from one another. Thus the process goals used to classify measures of effectiveness in Figure 2 can be easily adapted to describe the processes themselves (Figure 4). The distinction between analysis and construction process goals can apply directly to processes. That is, we hypothesize that analysis tasks differ sufficiently from construction tasks that, along with differences in the way they may be evaluated for effectiveness, there may also be different guidelines used in their construction. Thus figures 2 and 3 can also be mechanisms for identifying process model attributes. They should be accounted for in the process model as well as the effect on process.

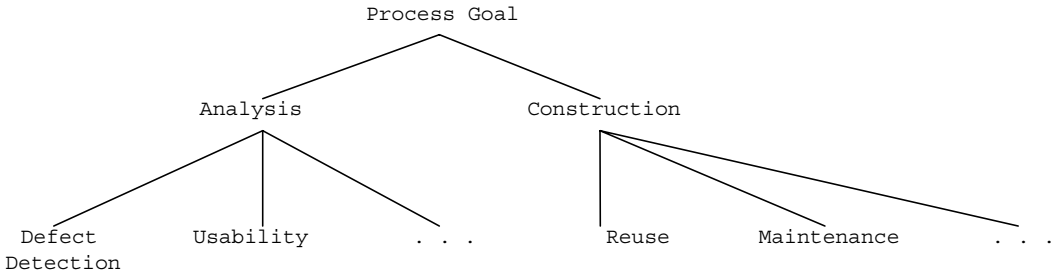

Figure 4: A portion of the hierarchy of possible values for describing the goal of a software engineering process.

Thus we can say that we are:

> analyzing a reading technique *for the purpose of* evaluating its ability to detect defects in a natural language requirements document

or we can say that we are:

> analyzing a reading technique *tailored to* defect detection in natural language requirements for the purpose of evaluation.

It depends on whether we are emphasizing the definition of the process or of its effectiveness.

In linking goal templates to hypotheses, we can think of the process model (object of study) as the independent variable, the effect on product (focus) as the dependent variable, and the context variables as the variables that exist in the environment of the experiment. The differences or similarities between experimental hypotheses can then be described in terms of these hierarchies of possible values. For example, consider the studies of DBR and PBR. In both cases, the process model was focused on the same

task (defect detection); although the notation differed, both were also focused on the same document (requirements). If all other attributes for process, product, and context models were held constant, we could begin to think of hypotheses at a higher level of abstraction. That is, instead of the hypothesis:

> **Subjects using a reading technique tailored to defect detection in natural language requirements are more effective than subjects using ad hoc techniques for this task**

The following hypothesis might be more useful:

> **Subjects using reading techniques tailored to defect detection in requirements are more effective than subjects using ad hoc techniques for this task.**

The difference between these hypotheses is that the focus of the study is described at a higher level of abstraction for the second hypothesis (requirements) than for the first (natural language requirements).

This difference in abstraction makes the second hypothesis more difficult to test. In fact, probably no single study could ever give us overwhelming evidence as to its validity, or lack thereof. Testing the second hypothesis would require some idea of what types of requirements notation are of interest to practitioners. Building up a convincing body of evidence requires the combined analysis of multiple studies of specific reading techniques for defect detection in requirements. But the effort required to formulate the hypothesis and begin building a body of evidence helps advance the field of software engineering. At best, the evidence can lead to the growth of a body of knowledge, containing basic and important theories underlying some aspect of the field. At worst, the effort spent in specifying the models forces us to think more deeply about the relevant ways of characterizing software engineering models that we, as researchers, are implicitly constructing anyway.

The above discussion should not be taken to imply that the attributes identified in Figures 1 through 4 are the only ones that are important, or for which hierarchies of possible values exist. To choose another example, in specifying the model of the context it is almost always important to characterize the experience of the subjects of the experiment. The most appropriate way of characterizing experience depends on many things; two possibilities are proposed in Figure 5.

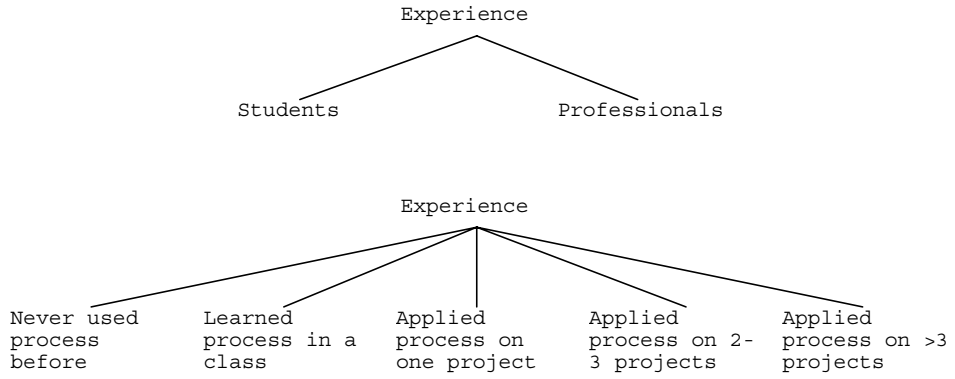

Figure 5: Two possible value hierarchies for measuring subject experience.

The trees shown in Figure 5 present two different ways of characterizing experience. The first is a simpler way of characterizing the attribute that distinguishes only between subjects who are still learning software engineering principles versus those who have applied them on real projects. The second hierarchy attempts to place finer distinctions on the amount of experience a subject has applying a particular process. Each may be appropriate to different circumstances.

4. Replicating Experiments

In preceding sections of this paper, we have tried to raise several reasons why families of replicated experiments are necessary for building up bodies of knowledge about hypotheses. Another reason for running replications is that they can increase the amount of confidence in results by addressing certain threats to validity: Internal validity defines the degree of confidence in a cause-effect relationship between factors of interest and the observed results, while external validity defines the extent to which the

conclusions from the experimental context can be generalized to the context specified in the research hypothesis [11]. 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 contributed to a growing body of knowledge on reading techniques.

4.1. 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.** Replications of this type vary none of the dependent or independent variables of the original experiment.
   - **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 internal threats to validity are addressed. For example, a replication may vary the order of activities to avoid the possibility that results depend not on the process used, but on the order in which activities in the experiment are completed.

     The attempt to compensate for threats to internal validity may also lead to other types of changes. For example, a process may be modified so that the researchers can assess the amount of process conformance of subjects. Although the aim of the change may have been to address internal validity, the new process should be evaluated in order to understand whether unanticipated effects on process effectiveness have resulted. Thus such a replication would fall into the second major category, discussed below.

2. **Replications that vary the research hypotheses.** Replications of this type vary attributes of the process, product, and context models but remain at the same level of specificity as the original experiment.
   - **2.1. Replications that vary variables intrinsic to the object of study (i.e. independent variables).** These replications investigate what aspects of the process are important by systematically varying intrinsic properties of the process and examining the results. This type of experiment requires the process 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. For example, researchers may question whether the specificity at which the process is described affects the results of applying the process. In this sense, the study of PBR2 may be seen as a replication of the study of PBR, in which the level of specificity of the process was varied but all other attributes of the process model remained the same.

   - **2.2. Replications that vary variables intrinsic to the focus of the evaluation (i.e. dependent variables).** Replications of this type may vary the ways in which effectiveness is measured, in order to understand for what dimensions of a task a process results in the most gain. For example, a replication might choose another effectiveness measure from those listed in Figure 2, investigating whether a defect detection process is more beneficial for finding errors than faults.

Other aspects of the focus model might be varied instead, e.g. a process might be evaluated on a document of the same type but different notation to see if it is equally effective (see Figure 3).

2.3. Replications that vary context variables in the environment in which the solution is evaluated. These studies can identify potentially important environmental factors that affect the results of the process under investigation and thus help understand its external validity. For example, replications may be run using the same process and product models as the original experiment but on professionals instead of students (see Figure 5) to see if the same results are obtained.

3. Replications that extend the theory. These replications help determine the limits to the effectiveness of a process, by making large changes to the process, product, and/or context models to see if basic principles still hold. We discussed replications in the previous category as replacing the value of some variable (e.g. document on which the process was applied, Figure 3) with another, equally specific value (e.g. SCR requirements instead of English-language requirements). Replications in this category, however, can be thought of as replacing an attribute of a process, product, or context model with a value at a higher level of abstraction (i.e. from a higher level in the hierarchy). Again using Figure 3, researchers may choose to study whether a type of process is applicable to requirements documents in general, rather than limiting their scope to a specific kind. The type of hypotheses associated with such replications was discussed in section 3.

4.2 Implications for Lab Package Design

In software engineering research, there has been a movement toward the reuse of physical artifacts and concrete processes between experiments. 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 [12], 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 4.1), 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. 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, in order to build up a body of knowledge about software engineering theories, researchers should know which experiments have been run that offer related results. 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 evidence relevant to basic theories. 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 facilitating the abstraction of results and experiences from series of well-designed studies. Interested readers are referred to existing examples of lab packages: [25, 26]. 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.

4.3. 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[4]. They have begun replicating experiments to better understanding the success factors of inspection and reading.

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.

5. Conclusions

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. Hypotheses that are of interest to the software engineering community and are written in a context that allow for a well defined experiment;
2. 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 context of experimentation;
3. A sufficient amount of information so that the experiment can be replicated and built upon; and
4. 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 in section 1, we can note that while it satisfied criteria 1 and 3, it failed with respect to criteria 2 and 4. 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.

In contrast, the set of experiments on reading, discussed in a working group at the 1997 annual meeting of

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

ISERN [18], 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 it 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 [11].

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.

It is our contention that interesting and relevant hypotheses can be identified and investigated effectively if empirical work is organized in the form of families of related experiments. In this paper, we have raised several reasons why such families are necessary:

- To investigate the effects of alternative values for important attributes of the experimental models;
- To vary the strategy with which detailed hypotheses are investigated;
- To make up for certain threats to validity that often arise in realistically designed experiments.

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 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.

Building up a body of knowledge from families of experiments has the following benefits for the software engineering researcher:

- It allows the results of several experiments to be combined in order to build up our knowledge about software processes.
- It increases the effectiveness of individual experiments, which can now contribute to answering more general and abstract hypotheses.
- It offers a framework for building relevant practical software engineering knowledge, organized around the GQM goal template or another framework from the literature.
- It provides a way to develop and integrate laboratory manuals, which can facilitate and encourage the types of replications that are necessary to expand our knowledge of basic principles.
- It helps generate a community of experimenters, who understand the value of, and can carry out, the needed replications.

The ability to carry out families of replications has the following benefits for the software engineering practitioner:

- It offers some relevant practical SE knowledge; fully parameterizing process, product, and context models allows a better understanding of the environment in which the experimental results hold.
- It provides a better basis for making judgements about selecting process, since practitioners can match their development context to the ones under which the processes are evaluated.
- It shows the importance of and ability to tailor “best practices”, that is, it shows how software processes can be altered by meaningful manipulation of key variables.
- It provides support for defining and documenting processes, since running related experiments assists in determining the important process variables.
- It allows organizations to integrate their experiences by making explicit the ways in which experiences differ (i.e. what the relevant process, product, and context models are) or are similar, and allowing the

abstraction of basic principles from this information.

Acknowledgements

This work was supported by NSF grant CCR9706151, NASA grant NCC5170, and UMIACS. The authors would like to thank Michael Fredericks and Shari Lawrence Pfleeger for their valuable comments on earlier drafts of this paper.

References

[1] V.R.Basili, "The experimental paradigm in software engineering", Experimental Software Engineering Issues: Critical Assessment and Future Directions, International Workshop, Dagstuhl, Germany, 1992. Appeared in Springer-Verlag, Lecture Notes in Computer Science, Number 706, 1993.

[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, pp.59-65, December 1996.

[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. 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.

[10] V. Basili, F. Lanubile, F. Shull, "Investigating maintenance processes in a framework-based environment", *Proc. of the Int. Conf. on Software Maintenance*, Bethesda, Maryland, pp.256-264, 1998.

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

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

[13] 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.

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

[15] C. M. Judd, E. R. Smith, and L. H. Kidder, *Research Methods in Social Relations*, sixth edition, Orlando: Harcourt Brace Jovanovich, Inc., 1991.

[16] 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.

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

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

[19] F. Lanubile, F. Shull, V. Basili, "Experimenting with error abstraction in requirements documents", *Proc. of the 5th Int. Symposium on Software Metrics*, Bethesda, Maryland, pp.114-121, 1998.

[20] 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.

[21] K. Popper, *The Logic of Scientific Discovery*, Harper Torchbooks, New York, NY, 1968.

[22] 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.

[23] 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.

[24] F. Shull. *Developing Techniques for Using Software Documents: A Series of Empirical Studies*. Ph.D. thesis, University of Maryland, College Park, December 1998.

[25] F. Shull, "Reading Techniques for Object-Oriented Frameworks", http://www.cs.umd.edu/projects/SoftEng/ESEG/manual/sbr_package/manual.html.

[26] F. Shull, "Lab Package for the Empirical Investigation of Perspective-Based Reading", http://www.cs.umd.edu/projects/SoftEng/ESEG/manual/pbr_package/manual.html.

[27] Z. Zhang, V. Basili, and B. Shneiderman, "An Empirical Study of Perspective-based Usability Inspection", Human Factors and Ergonomics Society Annual Meeting, Chicago, Oct. 1998.