# Building Knowledge through Families of Experiments

**Victor Basili, Forrest Shull, and Filippo Lanubile**[1]

## Abstract

Experimentation in software engineering is necessary but 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 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 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.

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.

**Keywords:** Empirical software engineering, experimental design, software process, software measurement, software reading techniques.

## 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[2]. 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 confidence not be placed in a theory unless it has stood up to rigorous deductive testing [34]. 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.

The scientific method has contributed to the progress of fields such as physics, medicine, and manufacturing. However, the method does not belong to any subset of intellectual endeavor: case studies of management

---

[1] Forrest Shull and Victor Basili are with the Computer Science Department, University of Maryland, College Park, MD 20742. E-mail: {fshull, basili}@cs.umd.edu. Filippo Lanubile is with the Dipartimento di Informatica, University of Bari, Via Orabona 4, 70126 Bari, Italy. E-mail: lanubile@di.uniba.it.

[2] For the purpose of this paper, we use the definitions of some key terms from [25] and [2]. 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 a causal relation among constructs of theoretical interest; the variables used to measure the casual construct are called *independent variables* while the variables used to measure the affected constructs are called *dependent variables*. 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.

information systems (MIS) can be as rigorous as experiments which are practiced in the natural sciences [29]. Unfortunately, in computer science and more specifically in software engineering, the balance between evaluation of results and development of new models is still skewed in favor of unverified proposals [45, 47]. The modeling research on software products, specifically models of program functions, has been mainly rooted in mathematics and there exists a large body of knowledge that can be used by developers. For other components, such as non-functional product characteristics, software processes, resources, there are also a fair number of models. But a body of evidence has not yet been built that enables a project manager to know with great confidence what software processes produce what product characteristics and under what conditions. Partly this is because of some features intrinsic to empirical work in these areas.

**Experimentation in software engineering is difficult.** Carrying out empirical work is complex and time consuming; this is especially true for software engineering. Unlike manufacturing, software developers 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.

The degree of credibility of any study depends on the validity of how conclusions are drawn. Campbell and Stanley have defined two classes of evaluation criteria: internal validity and external validity [15]. 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. Judd *et al.* add another class of evaluation criteria, construct validity [25], which defines the extent to which the variables successfully measure the theoretical constructs in the hypotheses. Finally, Cook and Campbell add one more class, conclusion validity [18], which defines the extent to which conclusions are statistically valid.

The difficulties intrinsic to software engineering (lack of data points, human factors) make it less likely that the validity types can all be satisfied at the same time: e.g. making a study more realistic to achieve a high external validity is in tension with the ability to manipulate the context to get a high internal validity. Still, 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 to 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. Other researchers should be aware of potential flaws or biases and their effects on the conclusions of a study. The opportunities for such flaws in empirical software engineering research are numerous: 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 [19].

**Drawing general 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 chief programming teams [1]);
- 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?).

Yet even when replications *are* run, it's hard to know how to abstract important knowledge without a framework for relating the studies.

To build a body of software engineering knowledge requires families of experiments, and a set of unifying principles that allows results to be combined and generalized. This requires a framework that makes explicit the different models used in the family of experiments and documents 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

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 the likely effects of these changes. Because an organizational framework provides a mechanism by which different studies can be compared, it helps to organize related studies and to tease out the true effects of both the process being studied and the environmental variables.

In this paper, we discuss a set of experiments performed by the authors and suggest an organizational framework for this set. We retroactively fit the experiments to this framework and discuss our experiences with software engineering experiments in the context of this framework. We show examples of the difficulty involved in running such experiments, but show how the framework can be used to better define experiments and combine them to overcome validity problems. We show how results might be unified and laboratory manuals generated to support the framework.

## 2. A Set of Experiments: Software Reading Techniques

Throughout this paper, we illustrate our framework and share our experiences with regard to a particular set of experiments on software 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 [3, 4]. 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 to a series of studies in which the following types of reading techniques were evaluated:

- Defect-Based Reading (**DBR**) focused on defect detection in requirements, where the requirements were expressed using a state machine notation called SCR [21, 36].
- Perspective-Based Reading (**PBR**) also focused on defect detection in requirements, but for requirements expressed in natural language [5].
- Use-Based Reading (**UBR**) focused on anomaly detection in user interfaces [48].
- Second Version of PBR (**PBR2**) consisted of new techniques that were more procedurally-oriented versions of the earlier set of PBR techniques. In particular, the techniques were more specific in all of their steps [40].
- Scope-Based Reading (**SBR**) consisted of two reading techniques that were developed for learning about an Object-Oriented framework in order to reuse it [11, 43].

## 3. The GQM Goal Template as a Tool for Experimentation

Several examples of organizational frameworks appear in the literature:

- [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). [26] 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.
- [30] 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.
- [19] presented 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.

Although any of the above would be helpful in this regard, for the purpose of this paper we find the Goal/Question/Metric (GQM) Goal Template [8] useful because it emphasizes the variables across which we are attempting to unify. 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, could be instantiated as:

> Analyze the development processes of a 1) disciplined-methodology team approach, 2) ad hoc team approach, and 3) ad hoc individual approach to characterize and evaluate with respect to cost and complexity (faults and rework) 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 more meaningful models? 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 of context variables).

For particular studies in this class, constructing a complete GQM template requires making explicit the models of process (object of study), effectiveness and product (focus), and context. Making these models explicit is necessary in order to understand what the study is testing as well as the conditions under which the empirical 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 processes might be classified first by scope as:

- Techniques (processes that require technical skill to accomplish some specific task),
- Methods[3] (processes that support techniques and are augmented with management issues such when and how a technique 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

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

knowledge builder in the context of a particular variable set

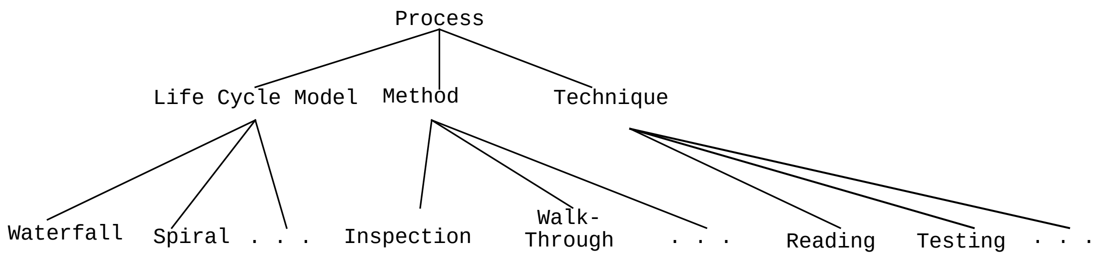


**Figure 1:** A portion of the hierarchy of possible values for describing software processes.

The reading technique experiments were concerned with studying the effect of the reading techniques 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 (Figure 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[4], or the amount of effort required, among other things.

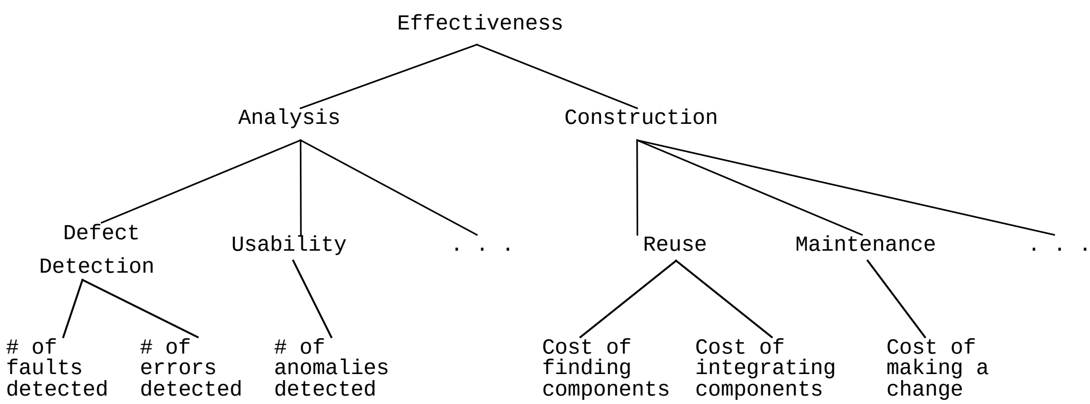


**Figure 2:** A portion of the hierarchy of possible values for describing the effectiveness of software processes.

[4] We use the following terms in a very specific way in this paper, based on the IEEE standard terminology [24]. 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 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.

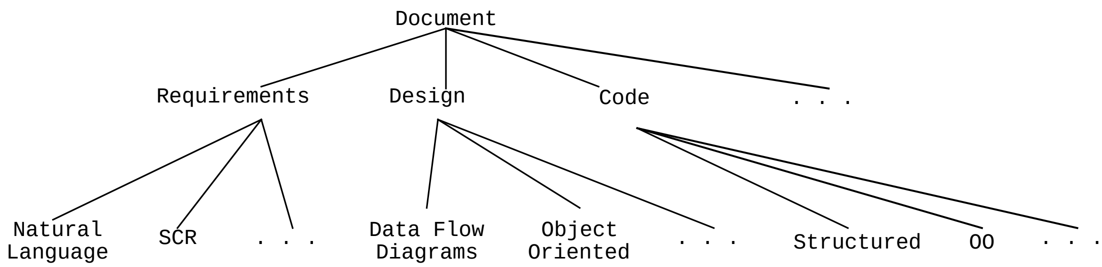


**Figure 3:** A portion of the hierarchy of possible values for describing software documents.

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 (Figure 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.

Thus, using the terminology in figures 1 through 3, the reading technique experiments can be described as follows:

- **DBR:** Analyze reading techniques to evaluate their ability to detect defects on SCR-notation requirements documents.
- **PBR:** Analyze reading techniques to evaluate their ability to detect defects on natural-language requirements documents.
- **UBR:** Analyze reading techniques to evaluate their ability to detect anomalies on screen shots of user interfaces.
- **PBR2:** Analyze reading techniques to evaluate their ability to detect defects on natural-language requirements documents. (Note that this GQM template does not capture the essential difference from the PBR experiment.)
- **SBR:** Analyze reading techniques to evaluate their ability to support reuse of Object-Oriented design and code.

In linking goal templates to hypotheses, the process model (object of study) can be thought of 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 the hierarchies of values for the model attributes. 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, hypotheses could be formulated 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 researchers and knowledge-builders are implicitly constructing anyway.

Instantiating these models is, by necessity, an iterative process. The value of an attribute for one model (e.g. process) may constrain the attributes of a second model (e.g. the product it is evaluated on), which may in turn constrain other attributes of the original model. However, in order to organize this paper we have dealt with each of the models in turn in the following sections. In each section, we use examples from our own experience to suggest concerns that must be taken into account when constructing the models. In particular, we will concentrate on identifying important attributes of each of the models, and the hidden dependencies that exist between the attributes. These attributes need to be measured, to the extent possible, by experimenters in order to fully describe the models of interest.

# 4. Modeling Process

Any process that is worthy of study should be of interest and value to the software engineering community. As such it should be definable as a set of steps, or better yet, bound by a set of attributes or properties. This means that experimenters have to articulate the steps of the process, and choose the attributes and their scope of values. In all of the reading technique studies, the techniques were meant to satisfy a certain set of properties, i.e., they were meant to be procedural, focussed, document and notation specific, and goal-oriented. Several of these properties are interrelated. In what follows, we discuss the difficulty and importance of dealing with the definition and scope of these attributes, based upon our experiences.

- **Procedural**

  The problem of how one defines a procedure to be followed and the level of detail to which it is specified is one of the most critical issues in the study of any process. If a process is specified procedurally, so that a user is guided through the steps of the process, then the process should be parameterized based on the level of specificity of the procedure. The level of specificity can be thought of as a spectrum of possible values, running from very detailed procedures in which every step is specified, to very high-level procedures in which only brief descriptions are given and the user selects activities based on his or her preferences and experiences. The appropriate level of specificity should vary from instance to instance and depends on what tradeoffs are most appropriate to the given environment. Should it be high-level, running the risk of providing too little support to the user? Should it be low-level, running the risk of over-specifying the process in a way that makes it tedious or impractical to follow? Should it vary with the experience of the user of the process? Besides a description of what the user should do, should there be a description of what should not be done?

  In our experience, these questions do matter. We have observed that there are differences in results for the same process, followed at different levels of specificity:

  *In the SBR study, subjects were given a process for reusing functionality from example applications. There were characteristic differences between subjects who followed the low-level and detailed process step-by-step and those who used a higher-level form of the process. Both advantages and disadvantages were noted. For example, subjects who followed the low-level process 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). Followers of the low-level process, however, also seemed less likely to waste time on unnecessary activities not called for by the process (they were less likely to get involved in "gold plating" the new system).*

  Moreover, our experiments have also provided some evidence that this process attribute needs to be tied to the context model (discussed in section 6) - in particular to the set of subjects who will be executing the process. In our experiment, we noticed a significant interaction between specificity and subject experience:

  *A hypothesis of the PBR2 study was that a more specific procedure would provide benefits to less experienced reviewers by providing more guidance. However, this was not found to be the case. Less experienced readers were less likely to have been exposed to the relevant concepts, and thus tended to find the very specific procedure confusing. Subjects with very high levels of previous experience in this area were also adversely affected by the greater level of detail, perhaps because it allowed them less opportunity to use their usual procedures. Only subjects in the middle range of experience (i.e. those who had been previously exposed to the concepts but had not applied them on many industrial projects) saw some benefit from the more specific* *procedures.*

- **Focused**
The reading techniques were meant to be focused, i.e. each specific technique should focus on just some aspect of the document rather than require the reader to be responsible for the entire document.

  *In PBR, specific techniques were focused on reviewing a document from the point of view of a particular user of the document, so that readers were restricted in their focus and not responsible for all conceivable defects. The aim was for the full set of techniques to provide coverage of the document. That is, although any one PBR technique represents only a particular way of viewing the document, the set of all PBR techniques should represent all of the important customers of the document, and therefore together uncover all defects of importance.*

  Focusing may or may not be done in a way that aims to achieve coverage of the entire document. For example, our experiences in the study of SBR were different:

  *In SBR, specific techniques were focused on different ways of learning enough about the system to support reuse. The goal of neither technique was to achieve coverage; the aim was not to learn about the entire system, but to save effort by learning just the portions that seemed promising for reuse.*

  This attribute is related to another attribute, the level of specificity in the procedure; more procedural techniques can be focused on aspects of a document with greater accuracy.

- **Document/notation specific**
All of the reading techniques discussed in this paper were tailored to both the document and notation. On the other hand, it is possible to imagine that some processes are not meant to be tailored so explicitly; e.g. a review process might be designed for use on a number of different documents, or for a particular document regardless of notation. Thus, deciding whether a process is meant to address a very specific class of document, or should be applicable to a larger set, is probably the first attribute that must be decided in tailoring a process for a particular use.

  Once a process model is made document-specific, the experimenters must evaluate the process on a document that matches the tailoring. Thus the value of this process attribute must match the product model, discussed in section 3.

  This attribute is highly related to the specificity with which the procedure is described; the level of specificity that can be achieved in a procedure is somewhat dependent on the amount to which the procedure can be tailored to a particular document and notation.

- **Goal-oriented**
Closer study of the process models for the reading techniques presented in section 2 would reveal that each technique is tailored to a specific task (e.g. defect detection, reuse of design) 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). We hypothesize that 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 Figure 2 can also be a mechanism for identifying process model attributes.

  This process attribute is related to the focus of the procedure. That is, whether or not the process goal can be achieved depends partly on the portion of the document on which the procedure is focused; if the procedure is focused on only a portion of the document, it must be ascertained whether that portion contains the information necessary to accomplish the task.

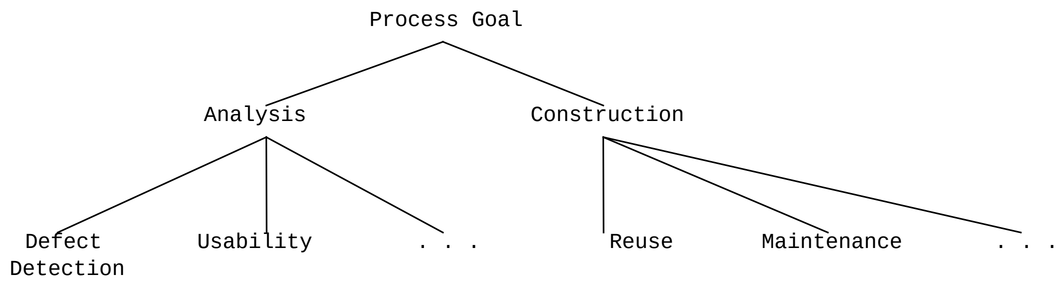


**Figure 4:** A portion of the hierarchy of possible values for describing the goal of a software engineering process.

The advantage of identifying the above set of attributes is that it helps us further specialize the model of the object of study (the process), and capture the essential differences and similarities of related models. For example, the following process descriptions summarize the objects of study in the reading technique experiments. To emphasize differences within the set of reading techniques, only some of the process model attributes are given:

- **DBR:** Analyzed three focused reading techniques tailored to detect defects in SCR-notation requirements documents, each using partial coverage of the document but whose union was meant to provide close to full coverage.
- **PBR:** Analyzed three focused reading techniques, defined at a low level of specificity, tailored to detect defects in natural-language requirements documents, each using partial coverage of the document but whose union was meant to provide close to full coverage.
- **UBR:** Analyzed three focused reading techniques tailored to detect anomalies of screen shots of user interfaces, each using partial coverage of the document but whose union was meant to provide close to full coverage..
- **PBR2:** Analyzed three focused reading techniques, defined at a high level of specificity, tailored to detect defects in natural-language requirements documents, each using partial coverage of the document but whose union was meant to provide close to full coverage.
- **SBR:** Analyzed two focused reading techniques tailored to reuse parts of an object oriented framework in the building of a new system, using partial coverage of Object-Oriented design and code.

Fully specifying the process models in such a way illustrates basic hypotheses for which evidence has been accumulated, e.g. does the level of specificity at which a procedure is described make a difference in its effects? Similarly, it also helps identify areas where hypotheses cannot be formulated as formally, which is usually an indication that more studies are needed:

*In the set of reading studies to date, we have noticed that analysis tasks (DBR, PBR, UBR, PBR2) seem to lend themselves particularly well to the union of partial coverage techniques, while construction tasks (SBR) require only partial coverage. Our current hypothesis is that analysis tasks typically require an entire document to be checked for multiple quality attributes, while construction tasks should aim to focus a reader's attention on only those aspects of a document suitable for the given task. More studies into construction tasks are needed before we can say whether this hypothesis can serve as a general guideline, or is merely a coincidence.*

# 5. Modeling Effectiveness on a Product

In this section, we discuss how a process is evaluated for effectiveness. As a necessary precondition, however, we first discuss process conformance: it is necessary that conclusions about a process are not drawn without knowing whether or not the process is actually being followed.

- **Process conformance**
  Our studies have emphasized that it should not be 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. There are two basic strategies for this: monitoring the execution of the process in order to understand the amount of

process conformance and reason about its effects (discussed in the following bullet), or attempting to specify the process in such a way that users must conform to the process. The latter strategy has been successfully used in several experiments in which it was desired to evaluate particular processes. One way of achieving this is to restrict the subjects' access to artifacts that are necessary for executing the process:

*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.*

A second alternative is to make the process include activities that are able to be checked, and that would require extra effort from the user if executed outside the process:

*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, it might be interesting to give subjects a very specific process and ask them to record many intermediate results, so that process conformance can be assessed at a very detailed level. However, this strategy suffers from the problem that it may make the process tedious to use and unpleasant for the subjects. The practical usefulness of such a technique should be questioned.

Yet another mechanism for helping 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 it was 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.*

Enforcing process conformance is not always the best strategy, as it may not reflect natural work practices. In [23], 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. When the object of the study is to gather information on process execution under more realistic conditions, it may be more appropriate to monitor rather than enforce process conformance:

*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.*

- **Validity of the measures**
  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 to define the intended effect, and how to measure it? Is the intended effect reduction in cost, improvement in quality? Can it be defined directly or by some metric that is easy to measure? A common problem is that the intended effect might not be fully understood, or cannot be defined sufficiently well to be measured. Therefore experimenters must measure indirectly using some metric that is felt to capture what is meant by effectiveness [20]. A common mistake is to choose a metric that is in reality not as well

correlated with our intended effect as assumed. It is necessary therefore to assess the construct validity of the experiment, i.e., 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 that reviewers discovered. However, is this the correct metric? The ultimate aim 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. [28]*

# 6. Modeling Context

The final model suggested by the GQM goal template is a model of the environment, i.e. any factors in the context in which the procedure is applied that may affect the results. This model is necessary to address the external validity of an experiment, i.e. how widely its conclusions can be extrapolated. Identifying important attributes of the context model is one of the largest challenges of empirical software engineering research; in this section, we present just a few examples of context factors and their effects on the manner in which experimental hypotheses can be explored..

- **Subject experience**
  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 the results will be generalized. 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, experimenters often cannot find enough subjects to achieve sufficient statistical power for testing group differences (see [12, 31] 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 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.

  However, it is not easy to identify the relevant skills or to measure experience in a way that is meaningful.

  *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. 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 no significant relationship between experience* *and number of defects found, that is, reviewers with more years of experience did not tend to find more defects than reviewers with less or no experience.*

A better way of approaching the issue of subject experience is to employ a mix of qualitative[5] 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:

*In the PBR study, 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, 39]. Experienced programmers, even with similar background, greatly vary with respect to their abilities. (It is expected 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 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.

This context attribute is related to many aspects of the process. As discussed in section 4, the appropriate level of specificity may be related to the experience of the users. The focus of a process may also depend on user experience, i.e. novices may be more effective when concentrating on different areas of a document than experienced users. This was the case in the focusing of the SBR techniques, also discussed in section 4.

- **Experimental context**
  Another important dimension is the **experimental context**, especially as it affects subject motivation. Software engineering process studies try to assess the impact of a technique on real work practices, and it is necessary for subjects to perform at a level that is representative of their professional work. Ideally an experimental setting would either reflect their organizational setting or would allow 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. (Thus this context attribute is very related to the product model; as the product model becomes more artificial, the context model differs increasingly from that which can be found in a realistic development environment.) 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 context”[6], 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.

---

[5] Qualitative data is information represented as words and pictures, not numbers [22]. 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) [38].

[6] The terminology “classroom context” 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.

Under normal circumstances in a classroom context, less process conformance is expected on the part of 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 when reviewing requirements (PBR or ad hoc) 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 a professional context 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. 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.*

The advantage of identifying the above set of attributes is that it helps us further specialize the context model and understand the limits of the experimental results. For example, Table 1 summarizes pertinent information about the context models for a number of different experiments that evaluated DBR. Information about the context models helps suggest new hypotheses about important environmental factors:

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, 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.

| Site | No. of Runs | Subjects | Results | Reference: |
|---|---|---|---|---|
| University of Maryland | 2 | Graduate students | Positive evidence | [36] |
| Lucent Technologies | 1 | Practitioners | Positive evidence | [35] |
| University of Bari | 1 | Undergraduate students | No evidence of increased effectiveness | [21] |
| University of Strathclyde | 2 | Undergraduate students | No evidence of increased effectiveness | [32] |
| University of Linkoeping | 1 | Undergraduate students | No evidence of increased effectiveness | [37] |

**Table 1:** Status of replications of DBR studies

# 7. Consequences for Experimental Designs and Threats to Validity

In previous sections we have looked at models of an empirical study's important components. In this section, we discuss briefly the experimental designs in which these models fit. This paper is not meant to be a primer for how to run experimental studies. Interested readers can find helpful guidelines for this task in [15, 25, 33], and more specifically for the software engineering domain in [20, 46]. However, in this section we discuss the impact of process, product, and context models on the kinds of designs that are feasible and useful for software engineering experimentation.

## 7.1. Experimental Design

As discussed in section 3, 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. The field simply lacks objective 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 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. In these cases, the comparison technique needs to be representative of currently accepted practices, so that the basis of comparison is well understood. This approach, while useful, typically suffers from a lack of a control group. This is because experiments usually rely on volunteers, and have to provide some benefit to the subjects and the organization that supports their participation. In cases in which subjects are students in a class, the experimenters 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 usual.

*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.*

In an ideal situation, an experimental design can be used similar to that seen in the Basili/Selby experiment [9], illustrated in Figure 5. All values of the controlled independent variable (i.e., the experimental treatments corresponding to the three processes being compared: code reading, functional testing, and structural testing) are experienced by each of the subjects. Such a within-subjects design mitigates the lack of a control group, because each subject effectively serves as his or her own control (i.e. any improvement for a subject can be measured against the baseline of his or her own previous performance). For the same reason, the variability among subjects' skills (discussed in section 6) is less likely to affect the results. Also, because each subject provides multiple data points the best use is made of the subjects. The order in which subjects encounter the treatments can also be varied 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 5:** Basili/Selby experimental design

Unfortunately, such a clean design is not always possible. The above design could be used in the Basili/Selby experiment because each process examined required different supporting artifacts, without which it could not be executed. Thus the experimenters could control when procedures were applied by the availability of the supporting documents. In other cases, certain attributes of the process model may prevent this approach from being used. Due to the level of specificity in the procedure (section 4) there are 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 procedurally defined technique (PBR) against a nonprocedural one. However, we were afraid that teaching the subjects a new and detailed procedure would bias later performance on the less procedural 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 less procedural, we considered it likely that some ideas from the new technique might find their way into the application of the comparison technique.*

A more specific procedure such as PBR, which provides instructions for what actions have to be carried out, might distort the later use of less procedural 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 specificity in the procedure. That is, teams should not be given the chance to apply less procedural techniques of their own after learning a more procedural technique, such as PBR. This prevents the less procedural techniques from incorporating guidance from the more procedural ones.

-

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

**Figure 6:** 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. 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 threat should be assessed in reference to subject experience (section 6) and is primarily a concern in classroom experiments; as discussed in section 6, 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 7 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 less to more procedural 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 7:** The experimental design of the PBR2 experiment.

### 7.2. Resulting Threats to Internal Validity

In previous sections of this paper we have discussed what we see as the very real constraints imposed by the nature of the software engineering discipline and empirical work. In section 7.1, we 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 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 in section 7.1 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 (Figure 5) is one of the few designs with no serious threats to internal 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 6) solves the practical problem of combining the execution of processes at varying levels of specificity 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. There could be a *learning effect*, especially of concern when subject experience is low, in which 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 there could be 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 – 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. The existence of a learning curve that increases reviewer effectiveness over time can be mitigated by providing training sessions before the actual treatments of the experiment. In this way, subjects are given the chance to overcome their learning curve during the training sessions rather than the actual experiment. Other aspects of the experiment can be manipulated to attempt to minimize the learning effect:

*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 was little to be learned from the first document that could be transferred to the second.*

Similarly, some of the other potential effects of time, such as boredom, can be mitigated 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 7 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 that can be done 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 there is 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 8. 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 more procedural process before the less procedural one.

| | 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 8:** Experimental design of DBR with reading technique varying between subjects.

# 8. Replicating Experiments

In this paper, we have raised several reasons why families of replicated experiments are necessary. This section discusses 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.

## 8.1. 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 conclusion 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 testing the same hypotheses as previous experiments have done, but altering the details of the experiment so that certain internal 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 [16] 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.*

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.

*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) allows us to understand the importance of the level of detail in reading techniques.*

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.

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, as discussed in section 5.

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 Table 1 and its discussion in section 6) 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).

*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.*

### 8.2. Implications for Lab Package Design

In software engineering research, there has been a movement toward the reuse of physical artifacts and 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, and to fine-tune those artifacts as more is learned. 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 [17], 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[7] to download experimental materials, either for reuse or for better understanding. In this way, these packages support replications of type 1.1 ("strict replications," as defined in section 8.1), 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 of type 1.2, which vary some aspects of the experimental setup. As discussed in section 7, the constraints imposed by the setting in which software engineering research is conducted mean that 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 of type 2, which 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 model must be explained in sufficient detail that other researchers can draw their own conclusions about key attributes. Since it is unreasonable to expect the original experimenters to determine all of the key attributes *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 product, effectiveness, and context models in sufficient detail that feasible changes can be identified and hypotheses made about their effects on the results. Sections 4 through 6 have given an overview of attributes that we have found to be of importance, but undoubtedly many others exist.

Finally, in order to address replications of type 3 and 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

[7] [14] distinguishes between internal replications (i.e, experiments repeated by the same researchers) and external replications (i.e., experiments repeated by researchers who are independent of the original experimenters). Lab packages are specifically designed to support and encourage external replications but they are valuable for internal replications too.

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 web pages can be updated continually, 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: [41, 42]. 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.

### 8.3. 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[8]. They have begun replicating experiments to better understand 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.

Finally it should be noted 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 [16]. 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 [44]. 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.*

## 9. Conclusions

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 (sections 4, 5, and 6);
- To vary the strategy with which detailed hypotheses are investigated (section 7.1);
- To make up for certain threats to validity that often arise in realistically designed experiments (section 7.2).

This discussion leads us to propose that the following are necessary before comprehensive bodies of knowledge can be built up 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

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

a well defined experiment;

2. Well-specified models of process, effectiveness, and context, in which key attributes can be changed in future studies to build up knowledge about important factors;
3. A sufficient amount of information so that the experiment can be replicated and built upon, varying the experimental design as necessary to address threats to validity; and
4. A community of researchers that understand experimentation and 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 ISERN [27], is an example that a body of knowledge has been built up by independent researchers working on different parts of the problem and exposing their conclusions to different plausible rival hypotheses. This set of experiments demonstrates that:

- The results of several experiments can be combined to build up knowledge about software processes.
- Techniques that are procedurally defined, document and notation specific, and goal driven, can be effectively designed and empirically validated for use.
- A procedural approach to a software engineering task can be more effective than a less procedural one under certain conditions (e.g., depending on experience, as discussed in section 4).
- A procedural approach to reading based upon specific goals will find defects related to those goals, so reading can tailored to the environment.

We have shown in this paper that experimental constraints in software engineering research make it very difficult, 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 all threats. 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 [15].

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 alter the values of key attributes of important models, 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 are addressed as a community. By running carefully designed replications, threats to validity in specific experiments can be addressed and evidence can be accumulated 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 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 software engineering knowledge; fully specifying 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 attributes.
- It provides support for defining and documenting processes, since running related experiments assists in determining the important process attributes.
- 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, Shari Lawrence Pfleeger, and the anonymous referees for their valuable comments on earlier drafts of this paper.

## References

[1] F. T. Baker, “Chief programmer team management of production programming”, *IBM Systems Journal*, vol.11, no.1, 1972.

[2] 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.

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

[4] 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

[5] 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.

[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, pp.256-264 .

[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] A. Brooks, J. Daly, J. Miller, M. Roper, and M. Wood, "Replication of experimental results in software

engineering", Technical Report, EFoCS-17-95 [RR/95/193], Dept. of Computer Science, University of Strathclyde, 1995.

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

[16] 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.

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

[18] T. D. Cook, and D. T. Campbell, *Quasi-Experimentation: Design and Analysis Issues for field settings*, Boston: Houghton Mifflin Co, 1979.

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

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

[21] 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.

[22] 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.

[23] G. Hidding, "Reinventing methodology", *Communications of the ACM*, vol. 40, no. 11, pp. 102-109, 1997.

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

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

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

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

[28] 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, pp.114-121.

[29] A. S. Lee, "A scientific methodology for MIS case studies", MIS Quarterly, March 1989, pp.33-50.

[30] 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.

[31] 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.

[32] 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.

[33] D. C. Montgomery, *Design and Analysis of Experiments*, 4th edition, John Wiley & Sons, 1997.

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

[35] A. Porter, and L. Votta, "Comparing detection methods for software requirements inspections: a replicated experiment using professional subjects", *Empirical Software Engineering Journal*, 3(4), 355-379, 1998.

[36] 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.

[37] K. Sandahl, O. Blomkvist, J. Karlsson, K. Krysander, M. Lindvall, and N. Ohlsson, "An extended replication of an experiment for assessing methods for software requirements inspections", *Empirical Software Engineering Journal*, 3 (4), 327-354, 1998 .

[38] C. B. Seaman, and V. R. Basili, "Communication and organization: An empirical study of discussion in inspection meetings", IEEE Transactions on Software Engineering, 24(6), June 1998.

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

[40] F. Shull. *Developing Techniques for Using Software Documents: A Series of Empirical Studies*. Ph.D. thesis,

University of Maryland, College Park, December 1998.

[41] F. Shull, “Reading techniques for Object-Oriented frameworks", http://www.cs.umd.edu/projects/SoftEng/ESEG/manual/sbr_package/manual.html.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] C. Wohlin, and P. Runeson (eds.), *Introduction to Experimentation in Software Engineering*, Technical Report, LUTEDX (TETS-7167), Dept. of Comm. Systems, Lund Inst. of Technology, Lund University, 1997.

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

[48] 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.

## Author Bios

Victor R. Basili is Professor of Computer Science at the University of Maryland, College Park, the Executive Director of the Fraunhofer Center - Maryland, and one of the founders and principals in the Software Engineering Laboratory (SEL) at NASA/GSFC. He works on measuring, evaluating, and improving the software development process and product. He is a recipient of a NASA Group Achievement Award and a NASA/GSFC Productivity Improvement and Quality Enhancement Award. He received the 1997 Award for Outstanding Achievement in Mathematics and Computer Science by the Washington Academy of Sciences. Dr. Basili has authored over 150 journal and refereed conference papers, has served as Editor-in-Chief of the IEEE TSE, and as Program Chair and General Chair of the 6th and 15th ICSE, respectively. He is co-editor-in-chief of the International Journal of Empirical Software Engineering, published by Kluwer. He is an IEEE and ACM Fellow.

Forrest Shull is a research associate in the Experimental Software Engineering Group at the University of Maryland. He has conducted empirical research into software engineering issues such as requirements reviews and Object-Oriented reuse. He has also worked in industry, developing software tools for companies performing research in biology, chemistry, and materials science. He received his Ph.D. degree from the University of Maryland in 1998. His current research interests include empirical software engineering, software reading techniques, software inspections, and process improvement.

Filippo Lanubile is an assistant professor of computer science at the University of Bari. He worked in the Experimental Software Engineering Group at the University of Maryland as a research associate. His research interests include experimental software engineering, software measurement, software inspection, framework-based development, and software evolution. He received a Laurea degree in computer science from the University of Bari. He is a member of the ACM and IEEE Computer Society.