The Role of Experimentation in Software Engineering: Past, Current, and Future

Victor R. Basili

Institute for Advanced Computer Studies and Department of Computer Science,
University of Maryland

Abstract. Software engineering needs to follow the model of other physical sciences and develop an experimental paradigm for the field. This paper proposes the approach towards developing an experimental component of such a paradigm. The approach is based upon a quality improvement paradigm that addresses the role of experimentation and process improvement in the context of industrial development. The paper outlines a classification scheme for characterizing such experiments.

1. Introduction

Progress in any discipline depends on our ability to understand the basic units necessary to solve a problem. It involves the building of models[1] of the application domain, e.g., domain specific primitives in the form of specifications and application domain algorithms, and models of the problem solving processes, e.g., what techniques are available for using the models to help address the problems. In order to understand the effects of problem solving on the environment, we need to be able to model various product characteristics, such as reliability, portability, efficiency, as well as model various project characteristics such as cost and schedule. However, the most important thing to understand is the relationship between various process characteristics and product characteristics, e.g., what algorithms produce efficient solutions relevant to certain variables, what development processes produce what product characteristics and under what conditions.

Our problem solving ability evolves over time. The evolution is based upon the encapsulation of experience into models and the validation and verification of those models based upon experimentation, empirical evidence, and reflection. This

[1] We use the term model in a general sense to mean a simplified representation of a system or phenomenon; it may or may not be mathematical or even formal.

encapsulation of knowledge allows us to deal with higher levels of abstraction that characterize the problem and the solution space. What works and doesn't work will evolve over time based upon feedback and learning from applying the ideas and analyzing the results.

This is the approach that has been used in many fields, e.g., physics, medicine, manufacturing. Physics aims at understanding the behavior of the physical universe and divides its researchers into theorists and experimentalists. Physics has progressed because of the interplay between these two groups.

Theorists build models to explain the universe - models that predict results of events that can be measured. These models may be based upon theory or data from prior experiments. Experimentalists observe and measure. Some experiments are carried out to test or disprove a theory, some are designed to explore a new domain. But at whatever point the cycle is entered, there is a modeling, experimenting, learning and remodeling pattern.

Science to the early Greeks was observation followed by logical thought. It took Galileo, and his dropping of balls off the tower at Pisa, to demonstrate the value of experimentation. Modern physicists have learned to manipulate the physical universe, e.g. particle physicists. However, physicists cannot change the nature of the universe [8].

Another example is medicine. Here we distinguish between the researcher and the practitioner. Human intelligence was long thought to be centered in the heart. The circulation of the blood throughout the body was a relatively recent discovery. The medical researcher aims at understanding the workings of the human body in order to predict the effects of various procedures and drugs and provide knowledge about human health and well-being. The medical practitioner aims at applying that knowledge by manipulating the body for the purpose of curing it. There is a clear relationship between the two and knowledge is often built by feedback from the practitioner to the researcher.

Medicine began as an art form. Practitioners applied various herbs and curing processes based upon knowledge handed down, often in secret, from generation to generation. Medicine as a field did not really progress, until various forms of learning, based upon experimentation and model building, took place. Learning from the application of medications and procedures formed a base for evolving our knowledge of the relationship between these solutions and their effects. Experimentation takes on many forms, from controlled experiments to case studies. Depending on the area of interest, data may be hard to acquire. However, our knowledge of the human body has evolved over time. But both grew based upon our understanding of the relationship between the procedures (processes) and its effects on the body (product). The medical practitioner can and does manipulate the body, but the essence of the body, which is physical, does not change. Again, the understanding was based upon model building, experimentation, and teaming.

A third and newer example is manufacturing. The goal of manufacturing is to produce a product that meets a set of specifications. The same product is generated, over and over, based upon a set of processes. These processes are based upon models of the problem domain and solution space and the relationship between the two. Here the relationship between process and product characteristics is generally

well understood. But since the product is often a man-made artifact, we can improve on the artifact itself, change its essence. Process improvement is performed by experimenting with variations in the process, building models of what occurs, and measuring its effect on the revised product. Models are built with good predictive capabilities based upon a deep understanding of the relationship between process and product.

2. The nature of the software engineering discipline

Like physics, medicine, manufacturing, and many other disciplines, software engineering requires the same high level approach for evolving the knowledge of the discipline; the cycle of model building, experimentation and teaming. We cannot rely solely on observation followed by logical thought. Software engineering is a laboratory science. It involves an experimental component to test or disprove theories, to explore new domains. We must experiment with techniques to see how and when they really work, to understand their limits, and to understand how to improve them. We must learn from application and improve our understanding.

The researcher's role is to understand the nature of processes and products, and the relationship between them. The practitioner's role is to build "improved" systems, using the knowledge available. Even more than in the other disciplines, these roles are symbiotic. The researcher needs ‘laboratories’; they only exist where practitioners build software systems. The practitioner needs to understand how to build better systems; the researcher can provide the models to make this happen.

Unlike physics and medicine, but like manufacturing, we can change the essence of the product. Our goal is to build improved products. However, unlike manufacturing, software is development not production. We do not re-produce the same object, each product is different from the last. Thus, the mechanisms for model building are different; we do not have lots of data points to provide us with reasonably accurate models for statistical quality control.

Most of the technologies of the discipline are human based. It does not matter how high we raise the level of discourse or the virtual machine, the development of solutions is still based upon individual creativity, and so differences in human ability will always create variations in the studies. This complicates the experimental aspect of the discipline. Unlike physics, the same experiment can provide different results depending on the people involved. This is a problem found in the behavioral sciences.

Besides the human factor, there are a large number of variables that affect the outcome of an experiment. All software is not the same; process is a variable, goals are variable, context is variable. That is, one set of processes might be more effective for achieving certain goals in a particular context than another set of processes. We have often made the simplifying assumption that all software is the same, i.e., the same models will work independent of the goals, context size, application, etc. But this is no more true than it is for hardware. Building a satellite

and a toaster are not the same thing, anymore than developing the micro code for a toaster and the flight dynamic software for the satellite are the same thing.

A result of several of the above observations is that there is a lack of useful models that allow us to reason about the software process, the software product and the relationship between them. Possibly because we have been unable to build reliable, mathematically tractable models, like in physics and manufacturing, we have tended not to build any. And those that we have, are not always sensitive to context. Like medicine, there are times when we need to use heuristics and models based upon simple relationships among variables, even if the relationships cannot be mathematically defined.

3. The available research paradigms

There are various experimental and analytic paradigms used in other disciplines. The analytic paradigms involve proposing a set of axioms, developing a theory, deriving results and, if possible, verifying the results with empirical observations. This is a deductive model which does not require an experimental design in the statistical sense, but provides an analytic framework for developing models and understanding their boundaries based upon manipulation of the model itself. For example the treatment of programs as mathematical objects and the analysis of the mathematical object or its relationship to the program satisfies the paradigm. Another way of verifying the results is by an existence proof, i.e., the building of a software solution to demonstrate that the theory holds. A software development to demonstrate a theory is different from building a system ad hoc. The latter might be an excellent art form but does not follow a research paradigm.

The experimental paradigms involve an experimental design, observation, data collection and validation on the process or product being studied. We will discuss three experimental models; although they are similar, they tend to emphasize different things.

First we define some terms for discussing experimentation. A hypothesis is a tentative assumption made in order to draw out and test its logical or empirical consequence. We define study broadly, as an act or operation for the purpose of discovering something unknown or of testing a hypothesis. We will include various forms of experimental, empirical and qualitative studies under this heading. We will use the term experiment to mean a study undertaken in which the researcher has control over some of the conditions in which the study takes place and control over (some aspects of) the independent variables being studied. We will use the term controlled experiment to mean an experiment in which the subjects are randomly assigned to experimental conditions, the researcher manipulates an independent variable, and the subjects in different experimental conditions are treated similarly with regard to all variables except the independent variable.

The experimental paradigm of physics is epitomized by the scientific method: observe the world, propose a model or a theory of behavior, measure and analyze,

validate hypotheses of the model or theory (or invalidate them), and repeat the procedure evolving our knowledge base.

In the area of software engineering this inductive paradigm might best be used when trying to understand the software process, product, people, or environment. It attempts to extract from the world some form of model which tries to explain the underlying phenomena, and evaluate whether the model is truly representative of the phenomenon being observed. It is an approach to model building. An example might be an attempt to understand the way software is being developed by an organization to see if their process model can be abstracted or a tool can be built to automate the process. The model or tool is then applied in an experiment to verify the hypotheses. Two variations of this inductive approach can be used to emphasize the evolutionary and revolutionary modes of discovery.

The experimental paradigm in manufacturing is exemplified by an evolutionary approach: observe existing solutions, propose better solutions, build/develop, measure and analyze, and repeat the process until no more improvements appear possible.

This evolutionary improvement oriented view assumes one already has models of the software process, product, people and environment and modifies the model or aspects of the model in order to improve the thing being studied. An example might be the study of improvements to methods being used in the development of software or the demonstration that some tool performs better than its predecessor relative to certain characteristics. Note that a crucial part of this method is the need for careful analysis and measurement.

It is also possible for experimentation to be revolutionary, rather than evolutionary, in which case we would begin by proposing a new model, developing statistical/qualitative methods, applying the model to case studies, measuring and analyzing, validating the model and repeating the procedure.

This revolutionary improvement oriented view begins by proposing a new model, not necessarily based upon an existing model, and attempts to study the effects of the process or product suggested by the new model. The idea for the new model is often based upon problems observed in the old model or approach. An example might be the proposal of a new method or tool used to perform software development in a new way. Again, measurement and analysis are crucial to the success of this method.

These approaches serve as a basis for distinguishing research activities from development activities. If one of these paradigms is not being used in some form, the study is most likely not a research project For example, building a system or tool alone is development and not research. Research involves gaining understanding about how and why a certain type of tool might be useful and by validating that a tool has certain properties or certain effects by carefully designing an experiment to measure the properties or to compare it with alternatives. An experimental method can be used to understand the effects of a particular tool usage in some environment and to validate hypotheses about how software development can best be accomplished.

4. Software engineering model building

A fair amount of research has been conducted in software engineering model building, i.e., people are building technologies, methods, tools, life cycle models, specification languages, etc. Some of the earliest modeling research centered on the software product, specifically mathematical models of the program function. There has also been some model building of product characteristics, such as reliability models. There has been modeling in the process domain; a variety of notations exist for expressing the process at different levels for different purposes. However, there has not been much experimenting on the part of the model builders: implementation yes, experimentation no. This may in part be because they are the theorists of the discipline and leave it to the experimenters to test their theories. It may in part be because they view their "models" as not needing to be tested - they see them as self-evident.

For example, in defining a notation for abstracting a program, the theorist may find it sufficient to capture the abstraction perfectly, and not wonder whether it can be applied by a practitioner, under what conditions its application is cost effective, what kind of training is needed for its successful use, etc. Similar things might be said about the process modeler.

It may also be that the theorists view their research domain as the whole unit, rather than one component of the discipline. What is sometimes missing is the big picture, i.e., what is the collection of components and how do they fit together? What are the various program abstraction methods and when is each appropriate? For what applications are they not effective? Under what conditions are they most effective? What is the relationship between processes and product? What is the effect of a particular technique on product reliability, given an environment of expert programmers in a new domain, with tight schedule constraints, etc.

One definition of science is the classification of components. We have not sufficiently enumerated or emphasized the roles of different component models, e.g., processes, products, resources, defects, etc., the logical and physical integration of these models, the evaluation and analysis of the models via experimentation, the refinement and tailoring of the models to an application environment, and the access and use of these models in an appropriate fashion, on various types of software projects from an engineering point of view. The majority of software engineering research has been bottom-up, done in isolation. It is the packaging of technology rather than the solving of a problem or the understanding of a primitive of the discipline.

5. What will our future look like?

We need research that helps establish a scientific and engineering basis for the software engineering field. To this end, researchers need to build, analyze and evaluate models of the software processes and products as well as various aspects of the environment in which the software is being built, e.g. the people, the or-

ganization, etc. It is especially important to study the interactions of these models. The goal is to develop the conceptual scientific foundations of software engineering upon which future researchers can build. This is often a process of discovering and validating small but important concepts that can be applied in many different ways and that can be used to build more complex and advanced ideas rather than merely providing a tool or methodology without experimental validation of its underlying assumptions or careful analysis and verification of its properties.

This research should provide the software engineering practitioner with the ability to control and manipulate project solutions based upon the environment and goals set for the project, as well as knowledge based upon empirical and experimental evidence of what works and does not work and when. The practitioner can then rely on a mix of scientific and engineering knowledge and human ingenuity.

But where are the laboratories for software engineering? They can and should be anywhere software is being developed. Software engineering researchers need industry-based laboratories that allow them to observe, build and analyze models. On the other hand, practitioners need to build quality systems productively and profitably, e.g., estimate cost track progress, evaluate quality. The models of process and product generated by researchers should be tailored based upon the data collected within the organization and should be able to continually evolve based upon the organization's evolving experiences. Thus the research and business perspectives of software engineering have a symbiotic relationship. From both perspectives we need a top down experimental, evolutionary framework in which research and development can be logically and physically integrated to produce and take advantage of models of the discipline that have been evaluated and tailored to the application environment. However, since each such laboratory will only provide local, rather than global, models, we need many experimental laboratories at multiple levels. These will help us generate the basic models and metrics of the business and the science.

This allows us to view our usable knowledge as growing over time and provides some insight into the relationship between software development as an art and as an engineering discipline. As we progress with our deeper understanding of the models and relationships, we can work on harder and harder problems. At the top is always the need to create new ideas, to go where models do not exist. But we can reach these new heights based upon our ability to build on packages of knowledge, not just packages of technologies.

6. Can this be done?

There have been pockets of experimentation in software engineering but there is certainly not a sufficient amount of it [5, 9, 11]. One explicit example, with which the author is intimately familiar, is the work done in the Software Engineering Laboratory at NASA/GSFC [6]. Here the overriding experimental paradigm has been the Quality Improvement Paradigm [1, 4], which combines the evolutionary

and revolutionary experimental aspects of the scientific method, tailored to the study of software. The steps of the QIP are:

Characterize the project and environment, i.e., observe and model the existing environment.

Set goals for successful project performance and improvement and organizational learning.

Choose the appropriate **processes** and supporting methods and tools for this project and for study.

Execute the processes, construct the products, collect and validate the prescribed data based upon the goals, and analyze it to provide real-time feedback for corrective action.

Analyze the data to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements.

Package the experience in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects and save it in an experience base for future projects.

To help create the laboratory environment to benefit both the research and the development aspects of software engineering, the Experience Factory concept was created. The Experience Factory represents a form of laboratory environment for software development where models can be built and provide direct benefit to the projects under study. It represents an organizational structure that supports the QIP by providing support for learning through the accumulation of experience, the building of experience models in an experience base, and the use of this new knowledge and understanding in the current and future project developments [2].

7. The maturing of the experimental discipline

In order to identify patterns in experimental activities in software engineering from the past to the present, I relied on my experience, discussions with the Experimental Software Engineering Group here at the University of Maryland, and some observations in the literature of experimental papers, i.e., papers that reported on studies that were carried out.

This identified some elements and characteristics of the experimental work in software engineering, specifically (1) identification of the components and purposes of the studies, (2) the types and characteristics of the experiments run, and (3) some ideas on how to judge if the field is maturing. These have been formulated as three questions. First, what are the components and goals of the software engineering studies? Second, what kinds of experiments have been performed? Third, how is software engineering experimentation maturing?

7.1. What are the components and goals of the software engineering studies?

Our model for components method is the Goal/Question/Metric (GQM) Goal Template [4]. The GQM method was defined as a mechanism for defining and interpreting a set of operation goals, using measurement. It represents a systematic approach for tailoring and integrating goals with models of the software processes, products and quality perspectives of interest, based upon the specific needs of a project and organization. However, here, we will only use the parameters of a goal to characterize the types of studies performed. There are four parameters: the object of study, the purpose, the focus, and the point of view. A sample goal might be: analyze **perspective based reading** (object of interest), in order to **evaluate** (purpose) it with respect to **defect detection** (focus) from the point of view of **quality assurance** (point of view). Studies may have more than one goal but the goals are usually related, i.e. there are several focuses of the same object being analyzed or a related set of objects are being studied. In experimental papers, the point of view is usually the researcher trying to gain some knowledge.

object of study: a process, product, or any form of 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 event?)

focus: 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: the person who benefits from the information, e.g., the researcher in understanding something better

In going through the literature, there appeared to be two patterns of empirical studies, those I will call *human factor* studies, and those that appear to be more broad-based software engineering. The first class includes studies aimed at understanding the human cognitive process, e.g., how individual programmers perceive or solve problems. The second set of studies appear to be aimed more at understanding how to aid the practitioner, i.e., building models of the software process, product, and their relationship. We will call these *project-based* studies. The reason for making the distinction is that they appear to have different patterns. Many of the human factor studies were done by or with cognitive psychologists who were comfortable with the experimental paradigm. The object of study tended to be small, the purpose was evaluation with respect to some performance measure. The point of view was mostly the researcher, attempting to understand something about programming.

Although the project-based studies are also often from the point of view of the researcher, it is clear that the perspectives are often practitioner based, i.e. the point of view represented by the researcher is that of the organization, the manager, the developer, etc. The object of study is often the software process or product in some form. If we are looking at breadth, there have been an enormous variety of objects studied. The object set which once included only small, specific

items, like particular programming language features, has evolved to include entire development processes, like Cleanroom development

Although the vast majority of such studies are also aimed at evaluation, and a few at prediction; more recently, as the recognition of the complexity of the software domain has grown, there are more studies that simply try to characterize and understand something, like effort distribution, rather than evaluate whether or not it is good.

7.2. What kinds of experiment have been performed?

There are several attributes of an experiment. Consider the following set:

(1) Does the study present results which are descriptive, correlational, cause-effect?

Descriptive: there may be patterns in the data but the relationship among the variables has not been examined

Correlational: the variation in the dependent variable(s) is related to the variation of the independent variable(s)

Cause-effect: the treatment variable(s) is the only possible cause of variation in the dependent variable(s)

Most of the human factor studies were cause-effect. This appears to be a sign of maturity of the experimentalists in that area as well as the size and nature of the problem they were attacking. The project-based studies were dominated by correlational studies early on but have evolved to more descriptive (and qualitative) style studies over time. I believe this reflects early beliefs that the problem was simpler than it was and some simple combination of metrics could easily explain cost, quality, etc.

(2) Is the study performed on novices or experts or both?

novice: students or individuals not experienced in the study domain

experts: practitioners of the task or people with experience in the study domain

There seems to be no pattern here, except possibly that there are more studies with experts in the project based study set. This is especially true with the qualitative studies of organizations and projects, but also with some of the controlled experiments.

(3) Is the study performed in vivo or in vitro?

In vivo: in the field under normal conditions

In vitro: in the laboratory under controlled conditions

Again, for project-based studies, there appear to be more studies under normal conditions (in vivo).

(4) Is it an experiment or an observational study? Although the term experiment is often used to be synonymous with controlled experiment, as defined earlier, I have taken a broader definition here. In this view, we distinguish between *experiments,* where at least one treatment or controlled variable exists, and *observational studies* where there are no treatment or controlled variables.

Experiments can be characterized by the number of teams replicating each project and the number of different projects analyzed. As such, it consists of four different experimental classes, as shown in Table 1: blocked subject-project, replicated project, multi-project variation, and a single project. Blocked subject-project and replicated project experiments represent controlled experiments, as defined earlier. Multi-project variation and single project experiments represent what have been called quasi-experiments or pre-experimental designs [7].

In the literature, typically, controlled experiments are in vitro. There is a mix of both novice and expert treatments, most often the former. Sometimes, the novice subjects are used to "debug" the experimental design, which is then run with professional subjects. Also, controlled experiments can generate stronger statistical confidence in the conclusions. A common approach in the blocked subject-project study is the use of fractional factorial designs. Unfortunately, since controlled experiments are expensive and difficult to control if the project is too large, the projects studied tend to be small.

Quasi-experiments can deal with large projects and be easily done in vivo with experts. These experiments tend to involve a qualitative analysis component, including at least some form of interviewing.

		# Projects	
		One	*More than one*
# of Teams per Project	***One***	Single Project	Multi-Project Variation
	More than one	Replicated Project	Blocked Subject-Project

Table 1: Experiments

Observational studies can be characterized by the number of sites included and whether or not a set of study variables are determined a priori, as shown in Table 2. Whether or not a set of study variables are predetermined by the researcher separates the pure qualitative study (no a priori variables isolated by the observer), from the mix of qualitative and quantitative analysis, where the observer has identified, a priori, a set of variables for observation.

In purely qualitative analysis, deductions are made using non-mathematical formal logic, e.g., verbal propositions [10]. I was only able to find one study that fit in this category and since it involved multiple sites would be classified as a Field Qualitative Study. On the other hand, there are a large number of case studies in the literature and some field studies. Almost all are in vivo with experts and descriptive.

7.3. How is software engineering experimentation maturing?

One sign of maturity in a field is the level of sophistication of the goals of an experiment and its relevance to understanding interesting (e.g., practical) things

about the field. For example, a primitive question might be to determine experimentally if various software processes and products could be measured and their characteristics differentiated on the basis of measurement. This is a primitive question but needed to be answered as a first step in the evolution of experimentation. Over time, the questions have become more sophisticated, e.g., Can a change in an existing process produce a measurable effect on the product or environment? Can the measurable characteristics of a process be used to predict the measurable characteristics of the product or environment, within a particular context? Can we control for product effects, based upon goals, given a particular set of context variables?

Another sign of maturity is to see a pattern of knowledge building from a series of experiments. This reflects the discipline's ability to build on prior work (knowledge, models, experiments). There are various ways of viewing this. We can ask if the study was an isolated event, if it led to other studies that made use of the information obtained from this particular study. We can ask if studies have been replicated under similar or differing conditions. We can ask if this building of knowledge exists in one research group or environment, or has spread to others, i.e., researchers are building on each other's work.

In both these cases we have begun to see progress. Researchers appear to be asking more sophisticated questions, trying to tackle questions about relationships between processes and product characteristics, using more studies in the field than in the controlled laboratory, and combining various experimental classes to build knowledge.

There are several examples of the evolution of knowledge over time, based upon experimentation and learning, within a particular organization or research group. The SEL at NASA/GSFC offers several examples [6]. One particular example is the evolution of the SEL knowledge of the effectiveness of reading related techniques and methods [3]. In fact, inspections, in general, are well studied experimentally.

		Variable Scope	
		defined a priori	*not defined a priori*
# of Sites	***One***	**Case Study**	**Case Qualitative Study**
	More than one	**Field Study**	**Field Qualitative Study**

Table 2: Observational Studies

There is also growing evidence of the results of one research group being used by others. At least one group of researchers have organized explicitly for the purpose of sharing knowledge and experiments. The group is called ISERN, the International Software Engineering Research Network. Its goal is to share experiences on software engineering experimentation, by experimenting, learning, remodeling and farther experimenting to build a body of knowledge, based upon empirical evidence. They have begun replicating experiments, e.g., various forms of replication of the defect-based reading have been performed, and replications of the per-

spective-based reading experiment are being performed. Experiments are being run to better understanding the parameters of inspection. ISERN has membership in the U.S., Europe, Asia, and Australia representing both industry and academia.

Another sign of progress for experimental software engineering is the new journal by Kluwer, the International Journal of Empirical Software Engineering, whose aim is to provide a forum for researchers and practitioners involved in the empirical study of software engineering. It aims at publishing artifacts and laboratory manuals that support the replication of experiments. It plans to encourage and publish replicated studies, successful and unsuccessful, highlighting what can be learned from them for improving future studies.

Acknowledgements: I would like to thank the members of the Experimental Software Engineering Group at the University of Maryland for their contributions to the ideas in this paper, especially, Filippo Lanubile, Carolyn Seaman, Jyrki Kontio, Walcelio Melo, Yong-Mi Kim, and Giovanni Cantone.

8. References

[1] Victor R. Basili, Quantitative Evaluation of Software Methodology, Keynote Address, First Pan Pacific Computer Conference, Melbourne, Australia, September 1985.

[2] Victor R. Basili, Software Development: A Paradigm for the Future, COMPSAC '89, Orlando, Florida, pp. 471-485, September 1989.

[3] Victor R. Basili and Scott Green, Software Process Evolution at the SEL, IEEE Software, pp. 58-66, July 1994.

[4] Victor R. Basili and H. Dieter Rombach, The TAME Project: Towards Improvement-Oriented Software Environments, IEEE Transactions on Software Engineering, vol. 14, no. 6, June 1988.

[5] V. R. Basili, R. W. Selby, D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, vol. SE-12, no. 7, pp. 733-743, July 1986.

[6] Victor Basili, Marvin Zelkowitz, Frank McGarry, Jerry Page, Sharon Waligora, Rose Pajerski, SEL's Software Development Process Improvement Program, IEEE Software Magazine, pp. 83-87, November 1995.

[7] Campbell, Donald T. and Julian C. Stanley, Experimental and Quasi-experimental Designs for Research, Houghton Mifflin, Boston, MA.

[8] Lederman, Leon, "The God Particle", Houghton Mifflin, Boston, MA, 1993

[9] Norman Fenton, Shari Lawrence Pfleeger, and Robert L. Glass, Science and Substance: A Challenge to Software Engineers, IEEE Software, pp. 86 – 94, July 1994.

[10] A. S. Lee, "A scientific methodology for MIS Case Studies", MIS Quarterly, pp.33-50 March 1989.

[11] W. L. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz, Experimental Evaluation in Computer Science: A Quantitative Study, Journal of Systems and Software, vol. 28, pp. 1

http://www.springer.com/978-3-540-24547-6

Foundations of Empirical Software Engineering
The Legacy of Victor R. Basili
Boehm, B.; Rombach, H.D.; Zelkowitz, M.V. (Eds.)
2005, X, 432 p., Hardcover
ISBN: 978-3-540-24547-6