The Experimental Paradigm in Software Engineering[1]

Victor R. Basili
Institute for Advanced Computer Studies
and
Department of Computer Science
University of Maryland

What is software and software engineering?

Software can be viewed as a part of a system solution that can be encoded to execute on a computer as a set of instructions; it includes all the associated documentation necessary to understand, transform and use that solution. Software engineering can be defined as the disciplined development and evolution of software systems based upon a set of principles, technologies, and processes.

We will concentrate on three primary characteristics of software and software engineering; its inherent complexity, the lack of well defined primitives or components of the discipline, and the fact that software is developed, not produced. This combination makes software something quite different than anything we have dealt with before.

One important characteristic about software is that it can be complex; complex to build and complex to understand. There are a variety of reasons for this. For example, we often choose software for a part of the solution, rather than hardware, because it is the part of the solution we least understand, or it is something new, or there is a requirement for change and evolution of the function or structure. In all of these cases complexity is introduced, the development becomes error prone, estimation is difficult, and there is a lack of understanding of implications of change.

However, the primary reason software is complex is probably the lack of models, especially tractable models of the product, process and any other forms of knowledge required to build or understand software solutions as well as the the interaction of these models. Software is not very visible, i.e., we do not have satisfactory models of the various aspects of the software, e.g., the functionality, the quality, the structure. In fact we do not even have intuitive models in many cases. This leaves us with a poor understanding of processes, requirements, and products.

Lastly, software is created via a development process, not a manufacturing process. This really means software is engineered.We have learned a great deal about quality manufacturing in the past few decades but we have not learned much about quality development/engineering.

So given the nature of this discipline, how does one begin to analyze the software product and process.

What are the available research paradigms?

There is a fair amount of research being conducted in software engineering, i.e., people are building technologies, methods, tools, etc. However, unlike in other disciplines, there has been very little research in the development of models of the various components of the discipline. The

[1] This paper appears in Experimental Software Engineering Issues: Critical Assessment and Future Directives, Proceedings of Dagstuhl-Workshop, edited by H. Dieter Rombach, Victor R. Basili , and Richard Selby,, September 1992, published by Springer-Verlag, #706, Lecture Notes in Computer Software, August 1993.

modeling research that does exist has centered on the software product, specifically mathematical models of the program function.

We have not sufficiently emphasized models for other components, e.g., processes, 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 research is mostly bottom-up, done in isolation. It is the packaging of a technology rather than the solving of a problem or the understanding of a primitive of the discipline.

We need research that helps establish a scientific and engineering basis for software engineering.

To this end, the research methodologies required involve the 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 organization, 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.

There are several example methodologies used in other disciplines [7]. These consist of various forms of experimental or analytic paradigms. The experimental paradigms require 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, the tend to emphasize different things.

1) The scientific method: observe the world, propose a model or a theory of behavior, measure and analyze, validate hypotheses of the model or theory, and if possible repeat the procedure.

In the area of software engineering this inductive paradigm might best be used when trying to understand the software process, product, people, 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. Two variations of this inductive approach are:

1.1) The engineering method: observe existing solutions, propose better solutions, build/develop, measure and analyze, and repeat the process until no more improvements appear possible.

This version of the paradigm is an evolutionary improvement oriented approach which 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.

1.2) The empirical method: propose a model, develop statististical/qualitative methods, apply to case studies, measure and analyze, validate the model and repeat the procedure.

This version of the paradigm is a revolutionary improvement oriented approach which 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. 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 is crucial to the success of this method. Proposing a model or building a tool is not enough. There must be some way of validating that the model or tool is an advance over current models or tools.

It is important to note that experimentation must be guided; there must be a rational for collecting data. Experiments must be designed to acquire information useful for the building of a suitable description (or model) of the systems under study. Experimentation alone is of no value if there is no underlying framework where experimental results can be interpreted. Other issues involved in these inductive, experimental methods include the types of experimental design appropriate for different environments, whether the experiment is exploratory or confirmatory, the validation of the data, the cost of the experiment, the problems of reproducibility, etc.

On the other hand, an analytic paradigm is:

2) The mathematical method: propose a formal theory or set of axioms, develop a theory, derive results and if possible compare with empirical observations.

This is a deductive analytical 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 their analysis of the mathematical object or its relationship to the program satisfies the paradigm.

Unfortunately, many projects and proposals that claim to be research are simply developments. These paradigms 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. The scientific 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.

How do you do experiments?

There are several different approaches to experimenting in the software engineering domain. One set of approaches can be characterized by the number of teams replicating each project and the number of different projects analyzed, as discussed in [6]. It consists of four different experimental treatments, as shown in Table 1: blocked subject-project, replicated project, multi-project variation, and single project case study.

The approaches vary in cost, level of confidence in the results, insights gained, and the balance between quantitative and qualitative research methods. Clearly, an analysis of several replicated projects costs more money but provides a better basis for quantitative analysis and can generate stronger statistical confidence in the conclusions. Unfortunately, since a blocked subject-project experiment is so expensive, the projects studied tend to be small. To increase the size of the projects, keep the costs reasonable, and allow us to better simulate the effects of the treatment variables in a realistic environment, we can study very large single project case studies and even multi-project studies if the right environment can be found. These larger projects tend to involve more qualitative analysis along with some more primitive quantitative analysis.

Because of the desire for statistical confidence in the results, the problems with scale up, and the need to test in a realistic environment, one approach to experimentation is to choose one of the treatments from below the line to demonstrate feasibility (statistical significance) in the small, and then to try one of the treatments above the line to analyze whether the results scale up in a realistic environment - a major problem in software engineering research.

Scopes of Evaluation

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

Table 1: ANALYSIS CLASSIFICATION: SCOPES OF EVALUATION

Given these research paradigms and experimental approaches, how do we apply them to the study of software and software engineering in practice? That is, software as an artifact does not exist in nature, it exists only where it is created. Therefore, it must be studied where it exists - in industrial and government environments. Under these circumstances, we need to provide proper motivation for organizations to allow researchers to "interfere" with their software development.

What is the relationship between software research and development in practice?

From a research perspective, we need to establish a scientific and engineering basis for software engineering. That is, we need industry based laboratories that allow us to understand the various processes, products and other experiences and build descriptive models understand the problems associated with building software, develop solutions focused on the problems, experiment with them and analyze and evaluate their effects, refine and tailor these solutions for continual improvement and effectiveness and enhance our understanding of their effects, and build models of software engineering experiences.

From a business perspective, we need to develop products and processes that will help us build quality systems productively and profitably, e.g., estimate the cost of a project, track its progress quantitatively, understand the relationships between models of cost, calendar time, functionality, various product qualities, etc., and evaluate the quality of the delivered product. These models of process and prooduct should be tailored based upon the data collected within the organization and should be able to continually evolve based upon the organizations evolving experiences.

That is, a successful business must understand the process and product, define process and product qualities, evaluate successes and failures, feedback information for project control via closed loop processes, learn from our experiences so we can do business better, package successful experiences so we can build competencies in our areas of business, and reuse successful experiences. It requires a closed loop process that supports learning, feedback, and improvement. Key technologies for supporting these needs include modeling, measurement, and the reuse of processes, products, and other forms of knowledge relevant to our business. This is, feedback loops are the development view of an experimental paradigm.

Thus the research and business perspectives of software engineering have a symbiotic relationship. That is, from both perspectives we need a top down experimental, evolutionary framework in which research and development can be focused, logically and physically integrated to produce and take advantage of models of the discipline, that have been evaluated and tailored to the application environment.

We can create experimental laboratories associated with the creation of software artifacts from which we can develop and refine models based upon measurement and evaluation and select the appropriate models to aid in development. Each such laboratory will help the business organization be more successful. However, since each such laboratory will only provide local, rather than global models or truths, we need experimental laboratories, at multiple levels. These will help us generate the basic models and metrics of the business and the science.

Can this be done?

There has been pockets of experimentation in software engineering but there is certainly not a sufficient amount of it [7]. One explicit example, with which the author is intimately familiar, is the work done in the Software Engineering Laboratory at NASA/GSFC [4]. Here the overriding experimental paradigm has been the Quality Improvement Paradigm [1,5], which combines the evolutionary and revolutionary experimental aspects of the scientific method, tailored to the study of software, i.e., the development of complex systems that need to have models built and evolved to aid our understanding of the artifact. It involves the understanding as well as the evolutionary and revolutionary improvement of software. The steps of the QIP are:

Characterize the current project and its environment.
Set the quantifiable **goals** for successful project performance and improvement.
Choose the appropriate **process** model and supporting methods and tools for this project.
Execute the **processes**, construct the products, collect and validate the prescribed data, 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 aid in the setting of goals, the guiding of the experiment, the rational for collecting data, the building of models, and the underlying framework for interpretation, the Goal/Question/Metric (GQM) paradigm was developed and had been evolving. The GQM is "a mechanism for defining and interpreting operational, measurable software goals. It combines models of an **object of study**, e.g., a process, product, or any other experience model, and one or more **focuses**, e.g., models aimed at viewing the object of study for particular characteristics that can be analyze from a **point of view**, e.g., the perspective of the person needing the information, which orients the type of focus and when the interpretation/information is made available for any **purpose**, e.g., to characterize, evaluate, predict, motivate, improve, specifying the type of analysis necessary to generate a **GQM model** relative to a **particular environment.**" [3]

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 for 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.

The Experience Factory concept supports a software evolution model that takes advantage of newly learned and packaged experiences, a set of processes for learning, packaging, and storing experience, and the integration of these two sets of functions. As such, it requires separate logical or physical organizations with different focuses/priorities, process models, expertise requirements. It consists of a **Project Organization** whose focus/priority is delivery and an **Experience Factory** whose focus is

"to support project developments by analyzing and synthesizing all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand. The **Experience Factory** packages experience by building informal, formal or schematized, and productized models and measures of various software processes, products, and other forms of knowledge via people, documents, and automated support." [2]

Via experimentation in the SEL, we have learned a great deal about the specific environment, as well as more general concepts such as: generating a multiplicity of process models, the relationship between process and product, measuring based upon goals and models The experimental nature of software development, defining closed loop processes for project feedback and corporate learning, defining and tailoring models, introducing new technologies into an environment, packaging and reusing a variety of experiences as models, evaluating experiences for reuse potential, supporting reuse oriented software development, and integrating packaged experiences.

There is a great deal more to learn. Similar activities need to be performed in a variety of environments and the resulting models form the various laboratories, or Experience Factories, need to by analyzed and compared to help us better understand the software and software engineering discipline and build the primitives of the science.

References

[1] V. R. Basili, "Quantitative Evaluation of Software Engineering Methodology," Proc. of the First Pan Pacific Computer Conference, Melbourne, Australia, September 1985 .

[2] V. R. Basili, "Software Development: A Paradigm for the Future", Proceedings, 13th Annual International Computer Software & Applications Conference (COMPSAC), Keynote Address, Orlando, FL, September 1989

[3] V. R. Basili, "The Goal/Question/Metric Paradigm", white paper, University of Maryland, 1990.

[4] V. R. Basili, G. Caldiera, F. McGarry, R. Pajerski, G. Page, S. Waligora, "The Software Engineering Laboratory - an Operational Software Experience Factory", International Conference on Software Engineering, May, 1992, pp. 370-381.

[5] V. R. Basili, H. D. Rombach "The TAME Project: Towards Improvement-Oriented Software Environments," IEEE Transactions on Software Engineering, vol. SE-14, no. 6, June 1988, pp. 758-773.

[6] V. R. Basili, R. W. Selby, Jr., "Data Collection and Analysis in Software Research and Management," Proc. of the American Statistical Association and Biomeasure Society Joint Statistical Meetings, Philadelphia, PA, August 13-16, 1984.

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