# Towards a Mature Measurement Environment: Creating a Software Engineering Research Environment

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

## Software Engineering Research

Software engineering researchers are building tools, defining methods and models. However, there are problems with the nature and style of the research. The research is typically bottom-up, done in isolation so the pieces cannot be easily logically or physically integrated. A great deal of the research is essentially the packaging of a particular piece of technology with little indication of how the work would be integrated with other pieces of research. The research is not aimed at solving the real problems of software engineering, i.e., the development and maintenance of quality systems in a productive manner. The research results are not evaluated or analyzed via experimentation or refined and tailored to the application environment. Thus, it cannot be easily transferred into practice. Because of these limitations we have not been able to understand the components of the discipline as a coherent whole and the relationships between various models of the process and product.

What is needed is a top down experimental, evolutionary framework in which research can be focused, logically and physically integrated to produce quality software productively, and evaluated and tailored to the application environment. This implies the need for experimentation, which in turn implies the need for a laboratory that is associated with the artifact we are studying. This laboratory can only exist in an environment where software is being built, i.e., as part of a real software development and maintenance organization. Thus we propose that Software Engineering Laboratory (SEL) type activities exist in all organizations to support software engineering research.

In this paper we will try to describe the SEL from a researcher's point of view. Jerry Page and Frank McGarry will discuss the corporate and government benefits of the SEL. I will try to focus my discussion on the benefits to the research community.

## The SEL as a Research Laboratory

The SEL is a laboratory that allows us to understand the various software processes, products and other experiences, build descriptive models of them, understand the problems associated with building software, develop solutions focused on the problems, experiment with the proposed solutions 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 relevant models of software engineering experiences.

The SEL has been in business for over 15 years and, based upon our experiences, its activities have evolved over time. In this section, I will describe the activities as they progressed over three phases.

The first phaseI will call the **understanding** phase because we worked on understanding what we could about the environment and measurement. During this period we measured what we could, used available models to explain the environment and our behavior, and built descriptive baselines and models typifying our environment.

In retrospect we made several mistakes. We collected too much data, i.e., because we did not know what was important we tended to collect all kinds of data hoping they would give us insights into the environment. We often blindly applied models and metrics without understanding the subtle assumptions and whether they were relevant in our environment. In a sense, we tried to evaluate things before we had built a deep understanding of what we were evaluating. We finally began to understand that measurement needed to be based upon models and goals. We established goals and a mechanism for generating measures based upon those goals, the first, primitive version of the Goal/Question/Metric Paradigm. This provided an informal approach to organizing our data. Based upon our goals, we began to build environment specific models by accumulating knowledge on individual projects and building baselines across multiple projects. Eventually we developed descriptive models that characterized the environment. These models included models of resources, defects, and product characteristics.

Once we had an understanding or characterization of the environment and the projects we were developing, we were able to begin the process of evaluation by comparing new projects against our baselines. This allowed us to proceed to phase two where the focus was on **improving** the process, product, and environment. During this phase, we continued to build up our data base of baselines and models, but we also evaluated and fed back information to the project. Many of these early data models were informal. The data was saved in a data base but the models existed mostly in documents. We began to experiment with various technologies to understand their effect, i.e. how they changed the baselines or the models we had. In order to provide a learning process across projects that would allow us to take advantage of what we had learned and evolve, we developed the Quality Improvement Paradigm, which is based upon an evolutionary, experimental approach to software improvement based upon both project and organizational feedback loops. The Goal/Question/Metric Paradigm continued to evolve to recognize different types of goals and questions and take advantage of the multi-project perspective. We began formalizing process, product, knowledge and quality models.

This need for formalization within the context of the Improvement Paradigm led to the concept of **packaging** models of our experiences so they were reusable on other projects. During this third phase we worked on choosing potentially reusable experiences, recognizing what was appropriate and relevant for the SEL. We began studying notations and mathematical formalisms for defining experiences.

There are several examples of current research projects in packaging experiences. For example, we are working on a project characterization model that allows us to recognize project patterns so that we can predict which projects look like the one we are working on. This allows us to package data for use as cost estimation models based upon our relevant past history [Briand, Basili, Thomas]. Having recognized that most experiences need to be modified for use, we have been defining models of tailorable experiences. For example, we are working on a tailorable test method [Basili, Martschenko, Swain]. The method allows one to choose the appropriate test techniques based upon the defect history of similar projects and the success rate of the techniques in that environment. Another example is the development of a model or reference architecture for different types of software factories [Basili, Caldiera and Cantone]. We are defining process models for reusing experience. We have developed a reuse- oriented evolution model [Basili and Rombach] and are working on integrating experience models [Oivo and Basili]. We have developed the concept of an Experience Factory, whose goal is to package software experiences and provide them to projects upon demand and have integrated the concept with an evolved QIP and GQM.

| | | |
|---|---|---|
| | | **Packaging** |
| | | SEL Ada Process |
| | | SEL Cleanroom Process |
| | | SME |
| | | Managers Handbook |
| | | Experience Factory |
| | **Improving** | |
| | Methodology Evaluation | Ada |
| | Cost Model Analysis | OOD |
| | Test Technique Analysis | Cleanroom |
| | QIP | CASE |
| **Understanding** | | |
| Modeling environment | Design Measures | Test Method |
| Data Collection (GQM) | Cost vs. Size Complexity | Reuse |
| Resource Baselines | | |
| Defect Baselines | | |

**Figure 1. Evolution of Measurement/Studies in the SEL**

Figure 1 represents some of the studies we performed and the hierarchy of the process, one phase based upon the other. That is, there was an understanding process (Phase 1), followed by an improving process (Phase 2), followed by a packaging process (phase 3). You can't improve until you understand, and you can't package until you can assess and improve. We are still understanding and trying to improve; these activities, along with packaging, will go on forever.

## The Research Framework Concepts

We have evolved to a framework [Basili b] that is based on three basic concepts, each of which is itself evolving:

o The Quality Improvement Paradigm (QIP), an evolutionary improvement paradigm, based upon the scientific method, tailored for the software engineering,

o The Goal/Question/Metric (GQM) paradigm, an approach for establishing project, corporate, and research goals and a mechanism for measuring against those goals,

o The Experience Factory, an organization that supports research and development by studying projects, developing and refining models, and supplying them to projects for further analysis and refinement.

The **Quality Improvement Paradigm** consists of the following steps:

o Characterize the current project and its environment with respect to a variety of models.

o Set the quantifiable goals for successful project performance and improvement.

o Choose the appropriate process model and supporting methods and tools for this project.

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

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

o 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 so it represents our current state of knowledge and is available for future projects.

The research emphasis is on taking each of these issues associated with the QIP, (e.g., characterizing, goal setting, choosing process, executing, analyzing, and packaging), and formalizing and integrating them. Each of these steps has evolved over the years. We have been building models of characterization. For example, what are good models that allow me to recognize what kind of software project I have and what projects are similar? Based on data, we are using pattern recognition techniques to recognize where to find the most appropriate kinds of experiences related to the current project [Briand, Basili, Thomas].

Goal setting has become a process of integrating models. A goal typically takes the form of analyzing some form of object from some perspective. I need models of both the object of study and the various perspectives of interest on that object.

We want to choose processes. A key issue here is that process is a variable; that I need to select, manipulate and change processes based on the characterization of the project and the environment and the goals established for this particular project.

Execution needs automated support. An automated system, SME, has been developed to support the accessing of data in a packaged form. The analysis and packaging issues are the major focuses of this paper.

The **Goal/Question/Metric Paradigm** is a mechanism for defining and interpreting operational and measurable software goals. Goals may be defined for any object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. A particular GQM model 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, such as models of cost, correctness, defect removal, changes, reliability, user friendliness, etc. This implies that there are models of these quality perspectives developed and available for use at anytime.

These models can be analyzed 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 of the information is made available and for any purpose, e.g., characterization, evaluation, prediction, motivation, improvement, which specifies the type of analysis necessary.

The result is a GQM model relative to a particular environment. Environments are distinguished based upon a variety of factors, e.g., problem factors, people factors, resource factors, process factors, etc.

## Experimental Approaches

Given a form of the scientific method, in the guise of the QIP, a mechanism to generate research hypotheses, in the guise of the GQM, what kinds of experimentation can we perform? The chart in Figure 2 offers four classes of studies that we can and have performed. The approaches can be characterized by the number of teams replicating each project and number of different projects analyzed.

|  |  | #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 |

**Figure 2. Classes of Studies and Scopes of Evaluation**

The single project case study is where most people begin. There is a project and someone has decided to study it. The results can provide some insight into project development in the environment.

A multi-project variation type study involves the measurement of several projects where factors, such as a method, can be varied across similar type projects. This allows the experimenter to study the effects of variations to the extent that the organization allows them to vary on different projects. In fact, that's literally what we do in the SEL. We have a large number of projects, we have standard baselines of how things should happen, and we start to perturb them by making changes and studying the effects of those changes.

The replicated project study involves several replications of the same project by different subjects. Each of the issues studied is applied to the project by several subjects but each subject applies only one of the technologies. It permits the experimenter to establish control groups.

The blocked subject-project study allows the examination of several factors within the framework of one study. Each of the issues studied is applied to a set of projects by several subjects and each subject applies each of the technologies under study. It permits the experimenter to control for differences in the subject population as well as study the effect of the particular projects.

There are two problems with the controlled types of experiments: (1) they are rather expensive and (2) if done for large pieces of software, for example, one year duration projects, they are hard to control, especially over several replications. Therefore, even though these types of experiments generate stronger confidence in the results than the non-controlled type experiments, they must be performed on small projects so the results do not scale up. If, however, these experiments are run on a small scale achieving reasonable statistical results, then there is motivation to experiment with the technologies on a larger scale in either a case study or a multi-project variation. Combining the results of the controlled experiment and the large- scale case study or multi-project variation, we can gain confidence in the validity of the experimental results.

It is clear in the SEL that we are avid believers in experimentation. We do not believe that any technology, method, tool, process model, etc. works under all circumstances. Everything has limits, areas where it works well or poorly. If we are dealing with technologies, we know they have limits. Experimentation is important in understanding those limits.

| **Single Project (Case Study)** | **Multi-Project Variation** |
|---|---|
| Independent V&V | Effect of Methodology |
| Cleanroom Process | Resource Model Studies |
| Defect Analysis Studies | |
| Ada/Object Oriented Design | |
| Code Reuse in Ada/Fortran | |

| **Replicated Project** | **Blocked Subject-Project** |
|---|---|
| Effect of Methodology | Reading vs. Testing |
| Cleanroom Process | |
| Ada/O-O Design | |

**Figure 3. Example Classes of Studies**

Figure 3 contains several example studies we have performed in the SEL. These studies cut across various experimental classes. When we have found something effective as a case study, we eventually turn it into a multi-project variation because it is effective for the environment.

## An Example Set of Studies

As an example of an effective process with which we have performed multiple types of experiments, consider the Cleanroom approach to software development, as suggested by Harlan Mills. We first ran a replicated project study at the University of Maryland that showed that the approach was very effective. We then decided to run a case study here in the SEL, which again was successful. We have since begun two new projects using the approach and will eventually have enough projects for an analysis based upon multi-project variation.

The key elements of the Cleanroom Process [Dyer], include a mathematically-based design methodology which includes: function specification for programs, state machine specification for modules, reading by stepwise abstraction, correctness demonstrations when needed, and top-down development. The implementation is done without any on-line testing by the developer. There is statistically-based, independent testing, based on anticipated operational use. Testing is done from a quality assurance orientation.

The replicated Cleanroom study had as its goals to evaluate the Cleanroom process with respect to its effects on the process, product and developers relative to differences from a non-Cleanroom process [Selby, Basili, Baker]. The experiment was run at the University of Maryland with 15 three-person teams,10 using Cleanroom. The project was an electronic message system (~ 1500 LOC). The teams were permitted 3 to 5 test submissions and the data collected consisted of background and attitude surveys, on-line activities of the developers, and test results.

The effect of the Cleanroom approach on the process was that the Cleanroom developers (1) felt they more effectively applied off-line review techniques, while others focused on functional testing, (2) spent less time on-line and used fewer computer resources, and (3) tended to make all their scheduled deliveries.

The effect of the Cleanroom approach on the product with regard to static properties was that the products developed using the Cleanroom approach had less dense complexity, a higher percentage of assignment statements, more global data, and more comments. With regard to operational properties, Cleanroom products more completely met requirements and had a higher percentage of test cases succeed.

Based on these results, we decided that it was worth running a case study in the SEL to see if the approach scaled up and how it worked with changing requirements. In applying the approach in the SEL, you will see an application of the QIP with regard to improving process. We begin with the characterization step which asks the question, "what relevant models exist that are available for reuse?" There were three models: the standard SEL model, which defines how software gets developed in the SEL in a

FORTRAN environment; the IBM FSD Cleanroom model that was applied on a prior project, and the experimental model we used for the replicated project.

The SEL goals were to characterize and evaluate the Cleanroom approach in general, and specifically with regard to changing requirements. In prior applications, Cleanroom had been used on projects where the requirements were basically fixed at the beginning of the study. One of the questions we were often asked after the replicated project study was "would this technology survive in an environment with changing requirements?" Since we had not experimented with changing requirements, we could not answer the question with much confidence.

What had been learned from the IBM/Cleanroom model application was the basic process model, methods and techniques and that the process very effective in the given environment. From the UoM/Cleanroom model application, we learned that no developer testing enforces better reading, the process is quite effective for small projects, formal methods are hard to apply and require skill, and there may be insufficient failure data to effectively measure reliability.

Based upon the existing models, our goals, and the lessons learned from prior applications of Cleanroom, we defined an initial SEL Cleanroom process model. We stole what was most effective from prior applications; for example, the training was consistent with the University of Maryland course and we emphasized reading by at least two reviewers.

Because this was a real project, and there was concern on the part of some of the developers about the effectiveness of reading, e.g., that you needed to test certain algorithms, we allowed back-out options, e.g., you could request permission to unit test certain types of algorithms. These back-out options were never used, but they did provide a comfort level for the developers. When we didn't know how to handle some aspect of the approach in this environment we applied the standard SEL process model as long as it didn't conflict in principle with what we were trying to do. We monitored and made changes to the process model in real-time. We wrote lessons learned, and we redefined the process for the next time out.

Some of the major positive results of the application of Cleanroom in the SEL include: the approach scales up to a 30,000 SLOC project, it can be used with changing requirements, productivity increased by about 30%, the failure rate during test reduced to close to 50%, there was a reduction in rework effort (95% of the fixes, as opposed to 58%, took < 1 hour to fix), only 26% of faults found by both readers (implying two readers are important), there were effort distribution changes, e.g., more time in design and 50% of code time spent reading, code appears in library later than normal and more like a step function, there was less computer use by a factor of 5.

Negative lessons learned include the fact that better training was needed for the methods and techniques. The kind of training we had at the university wasn't good enough. For example, we provided training where the examples were stacks, etc. This was not appropriate for the application. (One thing we have done on the second two Cleanroom projects is reuse parts of the first project as examples in the training.) We needed better mechanisms for transferring code to testers and the testers need to add requirements for output analysis of code. As expected, we did not have enough error data (with a 30,000 line project) to seed the reliability model so there was no payoff in reliability modeling in the SEL.

A side effect of this project was that it generated much more interest in improving the requirements. This requirements problem existed independent of Cleanroom, but the approach exposed the problem. So there has been a genuine push in having better defined requirements.

These results were for a 30,000 line project and a particular application. Is that the size limit for the Cleanroom process? Suppose we try a 100,000 line project ... what are the limits of this particular technology? When does it start to fall apart? Even if it doesn't work for a given size project, that's okay ... we now understand the bounds on that technology. It should not be expected that a technology works under all circumstances, every time, and every place. We have to understand as a community that technology has limits and that we have to select, and modify processes appropriate for the situation.

The next two experiments will emphasize the application of the formal models more, we are using the box structure approach, a change in the application domain for one project, and a scale up to a 100 KLOC for the other project.

This has been an example of the Quality Improvement Paradigm in terms of a particular process, and in terms of experimental design moving from controlled experiments to case studies in a real environment, and moving from case study to multi-project environment.

And we continue to evolve.

## Packaging the Experience

We have just discussed a form of packaging, the documentation of the Cleanroom process model. We currently have a working document that represents the model as we understand it today. But it will change as we learn!

Packaging experience requires the continual accumulation of evaluated experiences (learning) in a form that can be effectively understood and modified (experience models) into a repository of integrated experience models (experience base) that can be accessed and modified to meet the needs of the current project (reuse).

Systematic learning requires support for recording experience off-line generalizing and tailoring of experience formalizing experience. Off-line is a key word here. Packaging cannot be done as part of a project development. Someone cannot perform this analysis and build models at the same time they are building software. There needs to be a separate organization, either physically or logically separate.

Packaging useful experience requires a variety of models and an experience base. The models require formal notations that are tailorable, extendible, understandable, flexible and accessible. An effective experience base must contain accessible and integrated set of analyzed, synthesized, and packaged experience models that captures the *local* experiences.

The **Experience Factory** is a logical and/or physical organization (separate from the project organization) that supports project developments by analyzing and synthesizing all kinds of experience models acting as a repository for such experience supplying that experience to various projects on demand. It 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.

There are a variety of software engineering experiences that we can package: resource baselines and models, change and defect baselines and models, product baselines and models, process definitions and models, method and technique models and evaluations, products, lessons learned, quality models, etc. In the SEL, they exist in the form of standards, policies, tools. The documents range from sets of lessons learned to a manager's handbook.

There are many forms of packaged experience. We can use mathematical equations defining the relationship between variables, e.g., Effort = $a*Size^b$. We can present raw or analyzed data in the form of histograms or pie charts, e.g., % of each class of fault. We can plot graphs defining ranges of "normal", e.g., graphs of size growth over time with confidence levels. We can write specific lessons learned associated with project types, phases, or activities, e.g., reading by stepwise abstraction is most effective for finding interface faults, or in the form of risks or recommendations, e.g., definition of a unit for unit test in Ada needs to be carefully defined. We can create models or algorithms specifying the processes, methods, or techniques, e.g., an SADT diagram defining Design Inspections with the reading technique a variable dependent upon the focus and reader perspective.

For example, in the SEL we have a whole set of equations that define the relationships between a variety of variables [Basili, Panlilio-Yap]. Management can use these equations to understand, predict, and evaluate. In the SEL, example packaged relationships include:

Effort = 4.37 + 1.43devlines

Effort = 5.5 + 1.5newlines

Docpages = 99.1 + 30.9 devlines
Numruns = -108 + 151devlines
For projects under 50 KLOC we have:
Effort = .877 + 1.5newlines
while for projects over 50 KLOC we have:
Effort = 66.9 + .003 numruns
We have been able to demonstrate that methodology favorable impacts software cost and quality but cumulative complexity unfavorable impacts these factors [Basili a].

We have fault profiles that allow us to compare and analyze environments and projects. For example, what percent of faults of a particular type, based on a particular classification scheme, occur during a standard FORTRAN development. Are the percentages the same for an Ada development? We have been able to show that Ada reduces the percent of interface faults, but not by the amount one might expect based upon the ability of Ada compilers to check for interface faults [Brophy].

## Conclusions

Based upon our experiences, we need a set of experience factories or SELs, each focused on packaging local experiences by building and tailoring local models, integrating technologies, studying scale-up, building experience bases, and developing automated aids.

It is still hard to answer questions like: how big should an SEL be? should the experience factory only be domain specific, should it focus on a homogeneous environment?

If the SELs are focussed on homogeneous environments, we will need to integrate these local experience factories into a high level experience factory that abstracts from local experiences, looks for patterns across environments, and generates the basic models of the science. But how is this accomplished?

What we can do now is take advantage of the experimental nature of software engineering. Processes, products, and environments can be measured and can be used to support practical development and research. The integration of the Improvement Paradigm, the Goal/Question/Metric Paradigm, and the Experience Factory Organization can provide a framework for both development and research.

Based upon our experience, it helps us derive descriptive models of our experiences, understand our experiences and our problems, evaluate and learn from our experiences, and build effective prescriptive models of our experiences and our quality objectives. It can and should be applied today and evolve with technology.

Taking advantage of the experimental nature of software engineering has provided a winning situation for research and development. From a researcher's perspective the

SEL has been a smashing success. Its evolution has been slow, we have made many mistakes, but we have learned a lot.You don't have to make the same mistakes we did, you can learn from our experiences.

## References

[Briand, Basili, Thomas]
Briand, L.C., Basili, V. R., Thomas, W. M., A Data Analysis Procedure for an Effective Application of the Improvement Paradigm, University of Maryland Technical Report, March 1991.

[Basili](a)
V. R. Basili, "Can We Measure Software Technology: Lessons Learned from 8 Years of Trying," Proceedings of the Tenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, December 1985.

[Basili](b)
V. R. Basili, "Software Development: A Paradigm for the Future," Proc. 13th Annual International Computer Software & Applications Conference, Orlando, FL, September 20-22, 1989

[Basili, Martschenko, Swain]
Basili, V. R., Martschenko, W. N., and Swain, B. J., A Framework for Goal Directed Process Planning, University of Maryland, working paper.

[Basili, Caldiera and Cantone]
V. R. Basili, G. Caldiera, and G. Cantone, A Reference Architecture for the Component Factory, University of Maryland Technical Report CS-TR-2607, March 1991

[Basili, Panlilio-Yap]
V. R. Basili, N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," IEEE COMPSAC, October 1985.

[Basili, Rombach]
V. R. Basili and H. D. Rombach, "Support for Comprehensive Reuse," accepted for publication in Software Engineering Journal, IEE and British Computer Society, July 1991, and also UMIACS-TR-91-23, CS-TR-2606l, February 1991

[Dyer]
M. Dyer, "Cleanroom Software Development Method," IBM Federal Systems Division, Bethesda, Maryland, October 14, 1982.

[Oivo and Basili]
ES-Tame: A Prototype Implementation of the TAME Experience Base, University of Maryland, working paper.

[Selby, Basili, Baker]
R. W. Selby, Jr., V. R. Basili, and T. Baker, "CLEANROOM Software Development: An Empirical Evaluation," IEEE Transactions on Software Engineering, Vol. 13 no. 9, September, 1987, pp. 1027-1037.

[Brophy]
C. Brophy, "Lessons Learned in the Transition to ADA from Fortran at NASA/Goddard," UMIACS-TR-89-84, CS-TR-2305, August 1989