A Personal Perspective on the Evolution of Empirical Software Engineering

Victor R. Basili

University of Maryland
Fraunhofer Center for Empirical Software Engineering
basili@cs.umd.edu

Abstract This paper offers a four-decade overview of the evolution of empirical software engineering from a personal perspective. It represents what I saw as major milestones in terms of the kind of thinking that affected the nature of the work. I use examples from my own work as I feel that work followed the evolution of the field and is representative of the thinking at various points in time. I try to say where we fell short and where we need to go, in the end discussing the barriers we still need to address.

1 Introduction

I presented an earlier version of this work in a keynote at ISESE 2006 and published it in the Journal of the Brazilian Computer Society (JBCS) [1]. At the time I had been asked to offer a 40-year perspective on the evolution of empirical software engineering, from the past to the future. That was an arduous task. So I decided to simplify the task by making it a personal perspective, as I have worked in the field for 40 years. My hypothesis is that my work followed the evolution of the field. So, I offer my own opinions on how the field has evolved using mostly examples from my own work to support those opinions. I have some thoughts on how the field started, where we fell short, and where we need to go.

But first, I would like to discuss what makes Software Engineering uniquely hard to research, i.e., to build a body of usable knowledge for the discipline of software engineering [2]. Software engineering has several characteristics that distinguish it from other disciplines. Software is developed in the creative, intellectual sense, rather than produced in the manufacturing sense, and so the processes we need to study are development processes, not production processes. This unique aspect of the discipline, that each product is created rather than replicated, is probably the most important one, and greatly affects how we build models, evolve, and learn about the software discipline. It means that the context variables for different

software developments greatly affect how we develop software, i.e., there will always be variation in study results and we will never be able to control or maybe even identify all the context variables. The discipline creates a need for continual experimentation, as we explore how to modify and tailor processes for different environments, i.e., different sets of context variables.

One consequence of this is that process is a variable, goals are variable, and environment is a variable. That is, we need to select the right processes for the right goals for the environment we are analyzing. So, before we decide how to study a technique and its effects, we need to know something about the environment and the characteristics of the product we are about to build. The environment specifies the collection of context variables.

A second distinguishing characteristic of the software engineering discipline is software's intangibility, or one might say, the invisibility of its structure, components, and forms of development. This is compounded by a third characteristic, the field's immaturity, in the sense that we haven't developed sufficient models that allow us to reason about processes, products, and their relationships. These difficulties intensify the need to learn from the application of ideas in different situations and the requirement to abstract from what we see.

A final problem is that developing models of our experiences for future use (that is, reuse) requires additional resources in the form of money, organizational support, processes, people, etc. Building models, taking measurements, experimenting to find the most effective technologies, and feeding back information for corporate learning cost both time and money. These activities are not a by-product of software development. If these activities are not explicitly supported, independent of the product development, they will not occur and we will not make quality improvements in the development process. This turns out to be a major burden in the evolution of our understanding of the software engineering discipline. How is the large expanse of knowledge being captured, evolving with each new application, and being maintained in a form that is easy to integrate?

All this makes good experimentation difficult and expensive. Controlled experiments are expensive and can be confirmatory only in the small. They do not deal well with scale-up, the integration of one process with another, the understanding of the effect of context variables, etc. It also makes it difficult to build on past work and see where new work fits in the tapestry we are building of problem/solution bounds and limits.

So, let us discuss the evolution of empirical software engineering over the past four decades. I will try to characterize the nature of the discipline in each decade and map the changes across several *key variables*: the *kinds of studies* that were being performed, the set *of methods being used*, the nature of *publications*, the *community of researchers*, the status of *replications and meta-analysis*, and the role of *context variables*.

This article is organized in sections, each section representing a phase, roughly broken down into decades. Section 2 covers the early days (~ 1971 - 1979), running isolated studies for a particular purpose. Section 3 focuses on the building of

software process and technique knowledge in a single domain and environment, (~ 1980-1989). Section 4 deals with expanding our observations across environments by limiting the technologies being studied (~1990 – 1999). Section 5 focuses on tying different types of studies together to create some form of replication by taking advantage of different study types (~2000 – 2009). In each section, I will try to cover what I saw as the main changes in the approach that was introduced during that decade, giving related personal experiences within that decade and summarizing with a discussion of the key variables. Finally, Section 6 focuses on summarizing where I think we are, what we have learned, and the problems with progressing further.

2 Phase I: Isolated Studies (~ 1971 – 1979)

The very first software engineering experiment I was aware of was performed by Gerry Weinberg [33]. It was the genesis of a series of controlled experiments on the study of programmers. It was an interesting example of how people tried to follow the goals set out for them, e.g., code readability, code efficiency, etc. And when not given any advice, the self-imposed goal appeared to be performance.

These were the early days when researchers ran isolated independent studies for a particular purpose, using case studies or controlled experiments as the means to analyze a particular question of interest. It was a time when people were developing and using measures in general. The focus was on trying to identify an appropriate set of metrics. Many of us were learning about running an experimental study, and the need for baselines as a basis for evaluation. There were attempts to run a small number of controlled experiments but they were done mostly in isolation, not as part of a larger study.

Personal Examples: Two isolated studies I was involved in were the Iterative Enhancement product evaluation [9] and a methodology evaluation [6]. The motivations for the studies were specific to the work we had been doing. The former was a case study with Joe Turner where we used quantitative observations over time, measuring the product, and comparing the product with itself, using prior versions as baselines. The object of study was a compiler we were building for a family of languages [10]. This was a single isolated study aimed a demonstrating that a software product was improving using a particular measurement-driven incremental development approach. The latter was a controlled experiment analyzing the effects of a collection of methods centered on chief programmer teams, including structured design and structured coding. The experimental method applied was a replicated study (controlled experiment) with three treatments: teams using the methods, teams not using the methods, and single programmers, all performing the same task. The study, performed with Robert Reiter, was a single painstakingly designed study in a classroom environment using advanced software engineering students. The purpose was to identify an effective set of methods to use in our

software engineering class. These kinds of studies were rare but were typical of the state of the art.

Summary: With respect to our key variables, the *kinds of studies* were mostly in vitro controlled experiments analyzing the effects of a particular variable within one environment, typically with students, or a report on some in vivo measurement study of a project. The *publications* mostly consisted of project studies and reviews were mixed. It was hard to get controlled experiments published. Although I remember one published review of our controlled experiment which said "I already knew that methodology was good so what was the point of running the controlled experiment", even though the study won the TSE best paper award for that year. The *community of researchers* was very small with little or no interaction and consisted of mostly model builders, product metric developers, and some scattered set of individual experimentalists. The set of *methods* for experimental studies was mostly quantitative analysis, using nonparametric statistics. The *context variables* were taken as a given, not measured. There was no replication or meta-analysis.

3 Phase II: Multiple studies in ONE domain (~ 1980 – 1989)

This early work made it clear that experiments can be run in the software engineering domain that provide empirical support for various beliefs, insights into what and how to measure, evidence that we can use measurement to abstract what is occurring in software development. It stimulated the realization that experimentation and measurement were important aspects of software development and that the design of experiments is an important part of improvement (something Deming had been preaching in manufacturing for many years [20]), that evaluation and feedback are necessary for learning, and that we need to experiment with technologies to reduce risk, tailor the technique to the environment, and make improvements.

The study of the software engineering discipline is exploratory and evolutionary; it is an application of the scientific method. Controlled experiments are not always possible or useful in isolation, so we needed to focus more attention on informal exploratory studies using pre-experimental and quasi-experimental studies, i.e., experiments that lack the element of random assignment to treatment or control [19]. These less formal studies are more common in social science disciplines, like education, and can provide useful insights into the effects of processes on product characteristics in large projects. We should couple these informal, exploratory studies with more formal empirical studies such as controlled experiments, when possible, to provide more evidence that what we are observing is valid. This combination of methods takes advantage of what is possible to do given the nature of the software development discipline. It became clear to me at least, that the study of software engineering is a laboratory science requiring collaborating re-

search groups. Understanding the discipline requires exploratory study, confirmatory study if possible, identification and understanding of the effects of context variables, replications of various forms, and meta-analysis creating an integrated tapestry of information.

We need to take advantage of all opportunities we can find to explore various ideas in practice, e.g., test their feasibility, find out if humans can apply them, understand what skills are required to apply them, and test their interactions with other concepts. Based upon that knowledge, we need to refine and tailor each idea to the application environment in which we are studying it so it can be easily transferred into practice. We build and evolve models by trying out our ideas in practice and changing based upon what we have learned. Since the nature of the software engineering discipline is more exploratory than other disciplines, we are more dependent on the empirical application of methods and techniques.

Personal Example: The break from the mold of isolated studies for me was the development of the Software Engineering Laboratory (SEL) [13] at NASA Goddard Space Flight Center. The goals were to understand ground support software development for satellites and improve the process and product quality using observation, experimentation, learning, and model building [12].

In 1976, the idea of creating a laboratory environment to study software development was perhaps unprecedented. But it provided an excellent learning environment where potential solutions to problems were proposed, applied, and examined for their effectiveness evolving into more effective solutions. Characteristics that made this setup a good place for empirical research included the limited domain of the application, the use of professional developers, firm support from the local organization, the presence of a research team to interact closely with the practical developers, and a mix of developers and managers with different goals, personalities, and responsibilities. We created a consortium of the NASA dynamics group of managers and developers, the contractor (CSC) group of managers and developers, and the research group from the University of Maryland. Everyone participated in all aspects of the laboratory, i.e., managers and developers were part of the research team. The SEL was integrated into the overall activities of the organization and supported by the project budget, not the research budget. The balance created an environment with lots of feedback and collaboration. The original team that remained mostly throughout the SEL's existence were Frank McGarry (NASA), Jerry Page (CSC), Marv Zelkowitz, and myself (UMD).

The SEL lasted for 25 years (1976 – 2001) during which time we built baselines of various project variables (defects, effort, and project metrics) to better understand the environment and identify where better methods might make a difference. The focus moved to in vivo studies, collecting data from live projects, providing feedback from data collection and measures, and storing and analyzing large amounts of data. The work involved multiple projects and multiple methods in a *single environment and domain,* a strength at the time but a limit for developing broader knowledge across many context variables. The longevity of this work

allowed us to demonstrate order of magnitude improvements. Unfortunately, such longitudinal studies are very rare [5].

We learned a great deal not just by experiments, but by trying to understand the problems, applying potential solutions, and learning where they were successful and where they fell short. We ran controlled experiments and performed case studies, but they were done in the context of the larger evolutionary learning process [2].

We learned the importance of understanding the environment (recognizing which context variables in that environment were important), the need to build our own models to understand and characterize that environment (general models were too hard to parameterize for our environment due to the lack of broader context knowledge), the need to model the interactions among many variables (e.g., the environment, projects, processes, products), and that data collection has to be goal driven and well defined [11].

The learning process was more evolutionary than revolutionary. With each learning experience, we tried to package what we had learned into our models of the processes, products, and organizational structure.

The SEL used the university to test high-risk ideas. We built models and tested hypotheses. We developed technologies, methods, and theories as needed to solve a problem, learned what worked and didn't work, applied ideas that we read about or developed on our own when applicable, and all along kept the business going.

The SEL also allowed us to help create an empirical research community. Many students and visiting researchers spent years working in the laboratory, honing their empirical skills, contributing the knowledge and recognizing the need for collaboration. People like David Weiss, David Hutchens, Richard Selby, Dieter Rombach, Lionel Briand, Sandro Morasca, Carolyn Seaman, Filippo Lanubile, William Thomas, Forrest Shull, Manoel Mendonça, Guilherme Travassos, Jyrki Kontio, among others contributed greatly to the research activity.

The most important thing we learned was how to apply the scientific method to the software domain, i.e., how to evolve the process of software development in a particular environment by learning from informal feedback from the application of the concepts, case studies, and controlled experiments [4]. The informal feedback created the opportunity to understand where to focus our case studies and experiments. Informal feedback also, perhaps surprisingly, provided the major insights, i.e., interviews and informal discussion helped us discover important issues such as why a process was difficult to apply.

This work stimulated the realization that we need to package and integrate our experiences by building models and guidelines. Experience needs to be evaluated, tailored, and packaged for reuse so software processes must be put in place to support the reuse of experience. The SEL generated the concept of an experience base of packaged usable experience from the environment that could be used as a decision support system in the development of projects, but the experience packages defined were local to the SEL. The models could not necessarily be reused in

other environments but the mechanism for building and packaging the experiences could.

The SEL ran a workshop every year presenting what we had learned that year and requesting papers from others to present their work. This gave us a perspective on the state of the art and practice every year and allowed us all to share our experiences. At its peak, the SEL Workshop had audiences of over 300 attendees.

Summary: With respect to our *key variables*, the *kinds of studies* being run involved characterizing an environment via measurement (single environment, single domain), performing evaluations, building predictive models, making improvements. The *set of experimental methods* included more nominal and ordinal data and the use of pre-experimental, quasi-experimental studies and simply learning by the application of an idea. There were parametric models being built to capture and predict variables like cost and schedule [16]. The *publications* mostly consisted of project studies and reviews were mixed. For example, if your study should describe the limitation of some technique, the technique author was miffed, rather than appreciating what could be improved. The *community of researchers* was still small but beginning to grow. There was a metrics community of people trying to define product metrics that would predict defects and assess quality. The *context variables* began to be taken into account. There was no *replication* or *meta-analysis*.

4 Phase III: Tying Studies Together (~1990 – 1999)

During this time there were attempts to tie studies together. Controlled experiments, case studies, quasi-experiments, qualitative analysis were being used in various combinations, each useful in its own right for varying purposes. Controlled experiments were of value for identifying specific variable relationships while case studies provided the opportunity to scale up. We learned that you could reduce risk by running smaller experiments off-line using the mix of studies to build confidence in a theory based upon multiple treatments. Qualitative analysis began to play a major role in providing deeper insights into what was going on [28]. The major focus was on measuring the relationship between process and product. However, in our field, the kinds of studies performed and the topics studied were still dependent on the opportunities available.

For the 10th anniversary of TSE (1986), Rick Selby, Dave Hutchens, and I defined a framework for experimentation in software engineering and wrote a state-of-the-field paper recognizing that most of the papers in the literature dealt with either experimental studies of programmers in the small doing controlled experiments or data collection on projects in the large [7].

Personal Example: Harlan Mills [25] had defined a method for reading code called reading by stepwise abstraction. Based upon our need to improve quality in the SEL we decided to see if the technique would be effective. We made use of

the methodology template defined in [7] to study the effects of the approach using different experimental methods. Because the technique was quite new for the SEL environment, we first applied the approach in an advanced software engineering class at the university. We ran a fractional factorial controlled experiment design on 200 to 300 line code modules to check the effectiveness of the approach when compared to testing. The results were promising in that they showed that for certain types of defects, e.g., interface defects, the approach was more effective in uncovering those defects than functional testing. We found similar results on a replicated experimental design again with students on a 1000 lines of code project. This encouraged us to try the fractional factorial design with professional programmers on a set of 200 to 300 line modules. Here the results were even stronger, demonstrating the benefits of the approach. This allowed us to try it in a live project of about 40KLOC using internal NASA developers. The scale-up provided very positive results (e.g., significantly lower defect rates), encouraging us to test the approach on three other projects. The second study was again an internal NASA project of 22KLOC, with similar positive results. The third and fourth studies were larger scale and with the contractor as developer. The 160KLOC project did not show any improvement, so we modified the organizational communication on the fourth project of 140KLOC with positive results again [4]. The problem with the third study was that the contractor was to ask for help when they were confused about applying the approach. In the fourth project, we established bi-weekly meetings in which there was lots of discussion.

The mix of study approaches allowed us to gather information about the effects of the method as well as the effects of context variables (internal vs. contractor) and gain positive evidence about the effectiveness of the approach and how to improve it for a different context. The mix of study types on the same technique allowed us to build evidence that the technique could be effective across a set of context variables (project size, in-house vs. contracted out) and learn how to modify the environment for improvement of the technique.

Summary: With respect to our *key variables*, all kinds of *studies* continued to be performed on a variety of techniques and we began to see replications of some earlier experiments. There was sufficient research activity to create the Journal of Empirical Software Engineering (started in 1996) with the aim of publishing empirical studies, including replications, which were not previously *publish*able. Empirical research studies were still difficult to publish in a 10-page conference papers, as it was hard to cover all points in that limited page format. Reviewers were looking for more than could possibly be reported in ten conference pages. There was the realization that the evolution of the discipline required a *community of researchers* and teams performing studies. ISERN (started in 1993) created an opportunity for the growing international community of researchers to meet every year with the goal of supporting interaction and collaboration. Their goal was not to defend the need to experiment but to figure out how to do it better. Dieter Rombach created the Fraunhofer Institute for Experimental Software Engineering (1997), which aimed at working with companies to improve the software engi-

neering discipline, allowing Fraunhofer research the opportunity to interact with several different environments. The *set of methods* being used was broad, consisting of a mix of quantitative and qualitative studies, case studies, controlled experiments, and learning by application. C*ontext variables* were beginning to be taken seriously but not fully recognized as the important set of influencing variables that they are. *Replication* involved building some studies that varied the context, threats to validity; building knowledge across studies about a particular technology.

5 Phase IV: Expanding Studies across Domains and Environments (~2000 – 2009)

This period began to see the expansion of studies across domains and environments and the rise in collaborations. The overall focus became understanding the behavior of various processes in different contexts and environments, allowing for the creation of a decision support system that would provide organizations with support for selecting the right techniques for their particular context, domain, and environment based upon their ability to characterize them. The focus needs to be on specifying the effects of technologies, and experimentally identifying the effects, limits and bounds of techniques. So, technologists need to be more specific about what their techniques do and do not do and we need to evolve empirical evidence about various techniques, gaining new confidence over time by better understanding the effects of influencing variables. We need to concentrate on building a body of knowledge based upon empirical evidence.

This requires the gradual gathering of large amounts of information from different environments and the ability to identify the context variables that influence the outcomes for different environments. Clearly this implies a long-term set of studies, replications, and many collaborations, i.e., different groups need to collaborate to provide a sufficient range of domains and environments developing knowledge about the usefulness of various techniques in context. The work should contribute to an 'experience base' that accumulates the current state of our knowledge over time. Because of the size of the problem, we need to break the task into smaller parts, e.g., limiting the techniques studied or limiting the number of environments and domains, etc. There were several examples of building knowledge for a limited number of techniques in different environments and domains, i.e., studying the effect of context on those specific techniques.

Personal Example: I was involved in three such collaborations. The first one was the NSF-sponsored Center for Empirically Based Software Engineering (CeBASE): a consortium of the University of Maryland (UMD), the University of Southern California (USC), Fraunhofer CESE, Mississippi State (MSU), and the University of Nebraska at Lincoln (UNL) [30]. The CeBASE project goal was to enable a decision framework and experience base that would form the basis and

infrastructure needed to evaluate and choose among various software development technologies appropriate for the environment. CeBASE concentrated on a limited set of techniques, e.g., defect reduction techniques [16], such as reading, and COTS-based development approaches [3]. It also began to look at agile techniques. The research goal was to create and evolve an empirical research engine for building the research methods that could provide empirical evidence as to what works and under what conditions it works.

It was clear that there was a great deal of research required before we could comfortably build an empirical research engine that could be applied universally to evaluate and provide support for the appropriate use of evaluated methods. This research engine proposed by CeBASE involved defining and improving methods to

- Formulate evolving hypotheses regarding software development decisions
- Collect empirical data and experiences
- Record influencing variables (context)
- Build experience models in the form of lessons learned, heuristics/ patterns, decision support frameworks, quantitative models and tools
- Integrate models into a framework
- Test hypotheses by application
- Package what has been learned so far so it can be used and evolved

The results of the work were published in papers and slide presentations. An experience base was built which consisted of our experience with and advice about the use of various defect detection techniques and approaches to dealing with COTS products, in different environments [27]. It also contained the results of e-workshops where concepts were discussed and debated by experts in the areas of interest.

The idea of studying various techniques and maturing them through application to different environments to better understand the influencing context variables was continued as part of the NASA-sponsored High Dependability Computing Project (HDCP). Here the team again consisted of UMD, USC, and Fraunhofer CESE for the empirical work and a variety of universities for the development of the dependability techniques, e.g., Carnegie Mellon University CMU), University of Washington (UW), Massachusetts Institute of Technology (MIT), University of California Santa Barbara (UCSB).

Because the project was attempting to identify techniques to improve the dependability of the product, we built test beds to study, compare, and mature the techniques for practice. The test beds allowed us to minimize the risk of applying the techniques to live systems. Test beds developed were a simplified MARS Rover and a part of a tactical separation assisted flight environment. For example, the latter test bed was used to identify limits to a method and allow for the method developer to make improvements to the techniques based upon the empirical analysis [14]. The application of the techniques went from test beds to carefully moni-

tored projects to large-scale projects, allowing the techniques to evolve over use and provide the necessary information for the experience base. These test beds became part of the framework and needed to be maintained and evolved, an expensive proposition.

A third example of this knowledge building process is the work performed by the development time working group of the DARPA High Productivity Computing Systems (HPC) project where the domain is high-end computing [22]. The practical focus was improving the time and cost of developing high-end computing (HEC) codes and empirically evaluating the set of competitors for developing a new HPC machine. The research focus was on developing theories, hypotheses, and guidelines that allowed us to characterize, evaluate, predict, and improve how an HEC environment (hardware, software, human) affects the development of high-end computing codes. There was a large research team consisting of MIT Lincoln Labs, MIT, University of California San Diego (UCSD), UCSB, UMD, USC, Fraunhofer CESE, University of Hawaii (UH), MSU, UNL, and the San Diego Supercomputing Center (SDSC). Work proceeded by evolving a series of studies with novices and professionals using controlled experiments (grad students), observational studies (professionals, grad students), case studies (class projects, HPC projects in academia), surveys, and interviews (HPC experts).

Test beds varied from classroom assignments (Array Compaction, the Game of Life, Parallel Sorting, LU Decomposition, …) to compact applications (Combinations of Kernels, e.g., Embarrassingly Parallel, Coherence, Broadcast, Nearest Neighbor) to full scientific applications (nuclear simulation, climate modeling, …) being developed at California Institute of Technology (CalTech), Stanford University (SU), University of Chicago (UC), and University of Illinois (UIUC). As new knowledge was discovered, the results were stored in a publically available experience base. The content of the experience base was the empirical evidence collected in terms of the effects of various notations, e.g., MPI, Open MP, for different applications, classifications of high-end computing defects, and test beds. The contents also included experimental packages to support the running of experiments, such as checklists for instructors and experts running studies, instrumentation downloads, and data collection and analysis packages. Results were published in a variety of venues as well as stored in the experience base. But the experience base remained unmaintained. Some of the test beds were used by others for different purposes but they are becoming harder to find.

Each of the projects had a limited lifetime of only about three years. Why? Because a funding agency like NSF does not support long-term research endeavors; it felt it needed to continually identify new theories, techniques, and technologies. Unfortunately, there was not even follow-up funding to maintain the experience base that was developed. The CeBASE concepts formed the basis for the HDCP project, but the technique focus changed. Then NASA changed its focus from doing dependability research and again there was no support for maintaining that experience base. DARPA leadership changed and the decision was made that the companies would identify strengths and weaknesses of their own machines. The

wiki remains public http://hpcs.cs.umd.edu/ and http://hpcbugbase.org/ , but there is no maintenance for it.

In each of the examples, a great deal was learned but there was no support for maintaining the knowledge that was developed in a systematic, shareable, useable form. The evolution of a discipline like software engineering requires an experience base of knowledge on what to use and when across many environments and domains. There has not been sufficient support for maintaining the kind of decision support system that would help organizations build better software more efficiently.

One excellent example of a decision support system was the Clearinghouse project [32], whose goal was to capture experience for the DoD environment. A user would enter the best set of variables that described their environment and project and would be provided with whatever advice was available and the level of evidential support for that information. The project, like my own experiences above, died due to the time limit of the funding.

Summary: With respect to our variables, *studies* are being performed to evaluate techniques in multiple contexts and define the relationship between user needs and what's available. Journal and conference *publications* have come to expect some form of analysis from new methods, even if it is only a feasibility study. The community of researchers continues to grow; experimentalists are replicating each other's studies. There are numerous *repetitions* of a few experiments. The *set of methods* available became a rich palate of tools: a full mix of qualitative and quantitative methods, controlled and quasi-experiments, case studies, surveys, folklore gathering, structured interviews and reviews, etc. *Context variables* are being studied and characterized when possible. There are attempts to build knowledge across studies. A good example of the latter is the work of the SNT laboratory at the University of Luxembourg [18], where the focus is on model-based concepts and tools dealing with the set of problems associated with software validation and verification and on the improvement of software validation and verification activities in practice. The laboratory has several organizations as collaborators and a long-term focus.

6 Phase V: Now and the Future

To recapitulate, we can look at the evolution of the research in terms of the interplay of methods, context, and domain. Early work characterized the effects of various methods, (*all study variables fixed*) in isolation to address a particular problem. The desire for understanding broadened and baselines of various project variables (defects, effort, product and project metrics) were built within a single domain and context, identifying where methods might make a difference (*fixed context and domain, varied techniques)* e.g., SEL. Experimental work expanded to applying various experimental designs to examine a specific technique over a lim-

ited set of domains and context variables trying to broaden knowledge about the technique and minimize the threats to validity (*fixed technique, varied context and domain)*, e.g., reading technique studies. Then the research evolved to consider a limited set of techniques across several contexts and domains (*varied context to study context, fixed technique set),* e.g., CeBASE, and to quantitatively define the effects of various techniques that could solve a particular problem (*evaluate techniques for achieving particular goals and studying the relationships between both*), e.g., HDCP, where we identified the appropriateness and effectiveness of various methods to support the building of dependable systems under varying conditions before transferring them into practice (*introduced test beds (specific contexts) to study techniques)*. Finally, there was work on building knowledge in a particular domain, packaging that knowledge in an experience base so it can be used by others, demonstrating the effectiveness of various approaches and learning in what contexts they are effective (*fixed domain, studying techniques and context variables*) HPCS. We now return to a discussion of our key variables.

6.1 Kinds of Studies and Methods

With regard to the study of techniques, we see more papers, both conference and journals, containing a new idea, showing some kind of application, even it is only at the level of a feasibility study. This is in part due to journals and conferences requiring some form of data; i.e., it is clear that no technique should be published without trying it out first. This is a major change in the culture. But it should not end there; the feedback from the application should be used to identify the bounds and limits and open ideas for improvement. Unfortunately, the culture has not changed this much. Techniques need to be experimentally tested to see where they can be improved, even if we only 'learn by applying' as I like to call it [2]. We need to evaluate the bounds and limits of each technique and see how techniques can be integrated with others in the life cycle and what their integration buys you. There are several pockets of this kind of work and they are expanding all the time.

The collection of methods has expanded, including their integration into any particular study. There are many examples of building knowledge about the domain, identifying folklore and theories, doing ethnographic studies, interviews, and observations, building models using grounded theory, case studies, quasi-experiments, controlled experiments, and evolving models supported by evidence. We can find work testing models and hypotheses via studies of all kinds. The door is more open to this kind of research.

6.2 Community of Researchers

We have certainly evolved a community that talks and tries to work with each other. This year will be the 21st ISERN workshop and the number of members continues to grow; more importantly, ISERN keeps track of collaborations and there are many involving the exchange of graduate students and visiting researchers. But we need a more effective community collaboration and communication plan. Most young researchers are primarily interested in establishing themselves and their reputations. Getting a degree in an environment where there is already an established community is a great opportunity for them. Senior (tenured) researchers can afford to build the laboratory structure needed to do this type of work and to build collaborative groups. We need support for a living experience base that represents our combined and integrated experience, evidence, and knowledge at any point in time. This involves a well-defined collaborative research agenda.

I believe the discipline of software engineering will not move forward without such a collaborative research agenda, a community-supported living experience base, and a mature empirical study discipline.

6.3 Publications

The Empirical Software Engineering Journal (EMSE) is in its 18th year and its ISI impact rating has steadily grown. It has achieved an ISI rating of 1.854, second only by a tenth of a point to TSE, the top-rated SE journal. Papers are being submitted from a larger and larger collection of international researchers each year. The ACM/IEEE Empirical Software Engineering and Measurement Symposium ESEM (formally the International Symposium on Empirical Software Engineering ISESE before it joined forces with the Metrics Conference) is in its twelfth year. Journals like TSE welcome experimental work. So there are sufficient venues for publishing empirical research. We have textbooks that specialize in experimentation in the software engineering discipline, most notably, the second edition of 'Experimentation in Software Engineering' [34].

With regard to publications, the guidelines that exist are well defined [24] but are very long, especially for conference papers. So there is a need to supplement reports on a study with technical reports and web-based material that deals with all guideline issues. I believe journals are better than conferences as publication targets due to the feedback and dialog that is associated with the review process. The community needs to identify conference guidelines as to what must be included in the paper and what should be available in the technical report or on-line website. And they must identify ways to use various related publication forms to create an integrated whole.

Papers need to build on prior work. There is now a lot more literature around. Partly due to the history of isolated studies, we do not have a good enough culture of reading, referencing, and assimilating existing material. I admit to having been guilty of not identifying all the related references and integrating my work into the whole tapestry of results. For example, we have been criticized for a large number of studies on "inspections" that do not seem to recognize, build on, or integrate with the past work.

6.4 Context Variables

To me, covering context is the biggest problem and the reason why we need a very large community of researchers. There are too many influencing variables and we do not even know what they are or how to measure for them or the extent of their influence. They represent multidimensional categories such as subject experience, environment, domain, class of SE technologies applied. How many variables are hidden in these? If we are to be successful at building knowledge, we probably need to limit the scope of some of these categories for each research team, like focusing on specific domains, classes of technologies, or environments, expanding out slowly, unifying across the differences when possible. Of course, the problem then becomes integrating the limited scope studies of one group with the others in such a way that we can identify bounds on the extent of influence of the context variables so that a limited, useable set of models can be built.

6.5 Replications and Meta-Analysis

Building theories requires replication, varying the threats, varying the artifacts, and varying the population. These studies require coordination, collaboration, and independence. It takes a team to run an experiment; it is too hard to do it all alone. It involves multiple groups, multiple disciplines, and requires feedback on the design and discussion of the results. Replications require a level of independence, but I do not believe we are at a point where we can run a replication without some form of discussion with the earlier experimenters. We are not yet able to present all that needs to be covered in a conference paper or even a journal. The discussions after the fact are important to understand why the results are different and what that difference exposes about the subject or the study. This is where collaboration and communication are important. A subgroup of the ISERN community is collaborating on replications [23]. There is a workshop on replication (RESER) and there have been several attempts to coordinate studies. One of the original aims of EMSE was to publish replicated results and it has done so. This is real progress.

7 Concluding Remarks

We have come a long way in evolving the discipline of empirical software engineering, but we have a long way to go. Part of the reason for our slow progress has been the lack of an empirical culture within most Computer Science departments. They were mostly spawned by Mathematics departments and mathematics is not an empirical science. So we did not inherit an empirical mind set. The building of our research engine is at its infancy. It needs to be better understood not just by empiricists but by software engineers in general. Theoretical physicists understand and appreciate the work done by experimental physicists and use their results to evolve their own theories. This is not yet true in software engineering. Less than a decade ago, I paraphrased what a software engineer whose work I respect said to me: I am a smart guy and I know my technique is good, so why do I need experimental evidence? *Software engineering requires an empirical research engine that identifies the benefits, limits, and bounds of technologies*.

We need to build a tapestry of models and guidelines that represent our knowledge about the benefits, as well as the bounds and limits of techniques, methods, and life cycle models as well as models representing product characteristics of all kinds. The real question is: If I want a product to have certain characteristics, e.g., schedule achievement, minimum cost, reliability, correctness, safety, security, etc., what are the appropriate techniques, methods, and life-cycle models to achieve those characteristics? *Software engineering needs to codify the relationships between processes and products*.

If we are to make more progress in the discipline of software engineering in general, both in practice and research, that symbiotic relationship between practice and research has to be nourished so both groups can gain and the discipline can evolve. We need many applications of a process, taking place in different environments, each application providing a better understanding of the concepts and their interaction. Over time, all context variables need to be considered. Many of them will not even pop up until we have seen applications of the approach in practice by different people at different sites. *Empirical software engineering needs to balance the symbiotic relationship between theory and practice*.

Research teams need multiple forms of expertise, e.g., domain knowledge, software engineering knowledge, a variety of experimentation capabilities. I am always leery when reading a single-authored empirical paper. The team not only provides different levels of expertise but provides checks and balances on the study itself. *Empirical studies in software engineering need multi-disciplinary teams*.

I believe that replication plays the key role in software engineering. In this case, I do not mean the confirmation that a prior study's results were true or not, which is hard to do since it is hard to replicate the context of the prior study exactly, but 'replication' is needed to expand the context set in which the results may or may not be true and to understand why. This kind of replication requires close in-

teraction between the original study team and the replication team, because we cannot always communicate the original context variables. In a collection of replications of reading studies we did with several groups in Brazil, we found that we had a hard time capturing tacit knowledge in replicating experiments, even when the teams are collaborating [31]. *Replication in software engineering studies should be expanding knowledge rather than confirming it.*

The building of the tapestry of software engineering knowledge is too grand for any one group to perform. Empirical software engineering requires groups who share results in effective ways. They need a repository of evolving models and lessons learned that can be used, added to, and evolved by other researchers. For each group, the focus can be bounded, limiting the context, the domain, the collection of techniques, methods, and life cycle models studied. For example, we can build bodies of knowledge about specific domains. Then we can combine what has been learned from these domains to build larger bodies of knowledge across domains, understanding what is common and what is not. For each domain, this involves folklore gathering, interviews, case studies, controlled experiments, experience bases, etc. *Empirical software engineering needs to build collaborative, communicating communities.*

If we have begun to develop collaborative, communicating communities, what is still missing? First, the need to share results in a truly effective way requires a shared repository of evolving models and lessons learned that can be added to and used by researchers. Second is the requirement for long-term support across organizations and countries of collaborators, not an easy task. Third is the reward system for researchers. Academic researchers are rewarded for creating ideas and sharing them in papers. Sharing and collaborating in the way I am suggesting here takes lots of time, effort, energy, and financial support and may not always result in papers, at least in the beginning. Most disciplines build on each other's work by integrating with the results of work found in journal papers. It is a model that works for physics but I do not believe it works for software engineering. That is because many results of empirical studies are small and evolutionary and can only be truly evaluated based upon the comparison with how it affects the whole. The eventual knowledge base of the discipline is an interconnected and tightly integrated set of process and product characteristics that can only be built by collaborating communities. The result of this can be used as a decision support system integrating process effects with product needs. Medicine has been more successful in working toward this goal. *Empirical software engineering needs to build a decision support system / experience base that provides support of practice and an experience base which represents what we know about the discipline. The internal information is the same but the interfaces are different, geared to different populations.*

We need to understand the different roles of theory and experimentation. Is there one group that develops theory and another that does experimentation, like physics? I do not think so. Their roles are too tightly intertwined in software engineering. The feedback loop from theory to practice to theory is too intertwined and

requires rapid response times. There needs to be a desire on both sides to collaborate and a mechanism that supports it.

So building a discipline of software engineering is big science, requiring many collaborations with long-term goals and longitudinal studies, the development of a framework for communicating, coordinating, and integrating experiential models with long-term support that will exist and be available to capture all forms of evidence, like physics. We need methods that support the exploratory nature of this big science. The discipline cannot be understood only by analysis. We need to learn from applying the discipline whether relationships hold, how they vary, and what the limits of various technologies are, so we can know how to configure processes to develop software better. *Software engineering is big science*.

Acknowledgments Most of the work used here as personal examples was developed by many people collaborating as teams at the University of Maryland and its partner organizations. Members of the team are too numerous to mention and have varied over time. But I have had the good fortune to work with many exceptional people who should all be considered as co-authors of this paper. I thank Madeline Diep and Lionel Briand for giving me several suggestions to improve this paper.

References

1. V. Basili, The past, present, and future of experimental software engineering. J. Braz. Comp. Soc. [online]. 2006, vol.12, n.3,pp. 7-12. ISSN 0104-6500.
2. V. Basili, "Learning through Applications: The Maturing of the QIP in the SEL", in Making Software, Andy Oram and Greg Wilson, eds., O'Reilly, 2011.
3 .V. Basili and B. Boehm, COTS-Based Systems Top 10 List, IEEE Computer, vol. 34(5): 91-93, May 2001.
4. V. Basili and S. Green, "Software Process Evolution at the SEL," IEEE Software, vol. 11(4): 58-66, July 1994.
5. V. Basili, F. McGarry, R. Pajerski, and M. Zelkowitz, "Lessons Learned from 25 Years of Process Improvement: The Rise and Fall of the NASA Software Engineering Laboratory," Proceedings of the Twenty-Fourth International Conference on Software Engineering (ICSE), Orlando, FL, May 2002.
6. V. Basili, R. Reiter, R. Jr., A Controlled Experiment Quantitatively Comparing Software Development Approaches IEEE Transactions on Software Engineering, vol. 7(3): 299-320 (IEEE Computer Society Outstanding Paper Award), May 1981.
7. V. Basili, R. Selby, D. Hutchens, Experimentation in Software Engineering, IEEE Transactions on Software Engineering vol. 12(7): 733-743, July 1986.
8. V. Basili, R. Tesoriero, P. Costa, M. Lindvall, I. Rus, F. Shull, and M. Zelkowitz, "Building an Experience Base for Software Engineering: A report on the first CeBASE eWorkshop," Proceedings of the Product Focused Software Process Improvement Conference, Kaiserslautern, Germany, September 2001.
9. V. Basili, A. Turner, Iterative Enhancement: A Practical Technique for Software Development, IEEE Transactions on Software Engineering, vol. 1(4), December 1975.
10. V. Basili and A. Turner, "A Transportable Extendible Compiler," Software Practices & Experiences, vol.5 (3): 297-298, July-September 1975.

11. V. Basili and D. Weiss, “A Methodology for Collecting Valid Software Engineering Data,” IEEE Transactions on Software Engineering, vol.10(3): 728-738, November 1984.
12. V. Basili and M. Zelkowitz, Analyzing Medium Scale Software Development, in Proceedings of the Third International Conference on Software Engineering, May 1978.
13. V. Basili, M. Zelkowitz, F. McGarry, J. Page, S. Waligora, and R. Pajerski, “Special Report: SEL’s Software Process-Improvement Program,” IEEE Software, vol. 12(6): 83-87, November 1995.
14. Betin-Can, A., Bultan, T., Lindvall, M., Lux, B., & Topp, S. (2007). Eliminating synchronization faults in air traffic control software via design for verification with concurrency controllers. Automated Software Engineering, 14(2), 129-178.
15. B. Boehm and V. Basili, Software Defect Reduction Top 10 List, IEEE Computer, vol. 34(1): 135-137, January 2001.
16. B. Boehm and V. Basili, “Software Defect Reduction Top 10 List,” IEEE Computer, vol. 34(1): 135-137, January 2001.
17. B. Boehm, et al. Software Cost Estimation with COCOMO II, Prentice Hall, 2000.
18. Lionel C. Briand, "Embracing the Engineering Side of Software Engineering", IEEE Software 29(4): 96 (2012)
19. William R. Shadish, Thomas D. Cook, Donald T. Campbell, Experimental and Quasi-Experimental Designs for Generalized Causal Inference, Wadsworth Publishing, 2001.
20. W. Edwards Deming, Out of the Crisis (Cambridge, Massachusetts: MIT Press, Center for Advanced Engineering Study, 1986).
21. P. Donzelli and V. Basili, “A Practical Framework for Eliciting and Modeling System Dependability Requirements: Experience from the NASA High Dependability Computing Project,” Journal of Systems and Software, vol. 79(1): 107-119, January 2006.
22. L. Hochstein, T. Nakamura, V.R. Basili, S. Asgari, M.V. Zelkowitz, J.K. Hollingsworth, F. Shull, J. Carver, M. Voelp, N. Zazworka, P. Johnson, Experiments to Understand HPC Time to Development, Cyberinfrastructure Technology Watch Quarterly, vol.2(4A): 24-32, November 2006.
23. Natalia Juristo, Sira Vegas, “The Role of Non-Exact Replications in Software Engineering Experiments, Journal of Empirical Software Engineering 2011.
24. Barbara Kitchenham, et.al., Preliminary guidelines for empirical research in software engineering, IEEE Transactions on Software Engineering (TSE), Volume 28 Issue 8, August 2002, pages 721-734
25. Linger Mills and Witt, Structured Programming: Theory and Practice, Addison Wesley, 1979.
26. J. Maldonado, J. Carver, F. Shull, S. Fabbri, E.Dória, L.Martimiano, M. Mendonça, V. Basili, Perspective-Based Reading: A Replicated Experiment Focused on Individual Reviewer Effectiveness, Empirical Software Engineering: An International Journal, vol. 11(1): March 2006.
27. I. Rus, C. Seaman, M. Lindvall, V. Basili, and B. Boehm, “A Web Repository of Lessons Learned from COTS-Based Software Development,” Crosstalk, vol. 15(9): 25, September 2002.
28. C. Seaman and V. Basili, “Communication and Organization: An Empirical Study of Discussion in Inspection Meetings,” IEEE Transactions on Software Engineering, vol. 24(7): 559-572, July 1998.
29. William R. Shadish, Thomas D. Cook, Donald T. Campbell, Experimental and Quasi-Experimental Designs for Generalized Causal Inference, Wadsworth Publishing; 2 edition, January, 2001.
30. F. Shull, V. Basili, B. Boehm, A.W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M.Zelkowitz, “What We Have Learned About Fighting Defects,” Pro-

ceedings of the Eighth IEEE International Software Metrics Symposium, Ottawa, Canada, June 2002.
31. F. Shull, V. Basili, J. Carver, J. Maldonado, G. Travassos, M. Mendonca, and S. Fabbri, "Replicating Software Engineering Experiments: Addressing the Tacit Knowledge Problem," Proceedings of the First International Symposium on Empirical Software Engineering, Nara, Japan, October 2002.
32. Shull, F. and Turner, R., "An Empirical Approach to Best Practice Identification and Selection: The US Department of Defense Acquisition Best Practices Clearinghouse," Proc. ACM/IEEE International Symposium on Empirical Software Engineering (ISESE05), pp. 133-140. Noosa Heads, Australia, November 2005.
33. Weinberg, G. M. The psychology of Computer Programming, Van Nostrand Reinhold, New York, 1971.
34. C. Wohlin, P. Runeson, M. Hoest, M. Ohlsson, B. Regnell, and A. Wesslen, 'Experimentation in Software Engineering,' Springer, 2012.