The Maturing of the Quality Improvement Paradigm in the SEL

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

The Software Engineering laboratory uses a paradigm for improving the software process and product, called the Quality Improvement Paradigm [Ba85, BaRo88]. But this paradigm has evolved over the past 18 years, along with our software development processes and product. Since 1976, when we first began the SEL, we have learned a great deal about improving the software process and product, making a great many mistakes along the way. For example, we tried to assess the quality of our processes and products before we understood what they were. When trying to understand, we were data driven rather than goal and model driven. We tried to use other people's models to explain our environment rather than recognizing we had to build models of our own environment before we could compare it with others.

The learning process has been more evolutionary than revolutionary. We have generated lessons learned that have been packaged into our processes, products and organizational structure over the years. We have used the SEL as a laboratory to build models, test hypotheses. We have used the University to test high risk ideas and develop technologies, methods and theories when necessary. We have learned what worked and didn't work, applied ideas when applicable and kept the business going with an aim at continually improving and learning.

This paper offers a personal perspective on how our approach to quality improvement has evolved over time and where I think we are evolving. I will try to carry you through various phases of our evolutionary learning process, arbitrarily breaking the learning into five year periods. showing you some of the things we did wrong and what caused us to change our ideas. I will use the Quality Improvement Paradigm steps themselves, as it presently stands, as a guidelines to how our thinking evolved based upon experiences in the SEL.

But first, let me give you the Quality Improvement Paradigm, as it is currently defined. In its full version, it can be broken up into six steps:

1. **Characterize the** current **project and** its **environment** with respect to the appropriate models and metrics.

2. **Set** the quantifiable **goals** for successful project performance and improvement.

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

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

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

6. **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 to be reused on future projects.

We often use a shortened version of the paradigm which is defined as three steps: understand, assess, and package. These steps can be mapped onto the six steps by noting that understand is step 1, assess is steps 2 through 5 and package is step 6.

Each of these steps changed over time, either in how we defined them or how we implemented them. Characterization went from collecting metrics to defining baselines to building models. Goal setting started out as simply data collection, evolved to being goal driven and finally goal and model driven, i.e., data collected based upon goals and quantifiable models. The processes, methods and technologies available in the process selection step evolved from combinations of heuristic methods, to well-defined technologies, to high impact, combinations of integrated technologies, methods, and life cycle models, to the evolving and tailoring processes to the situation. During process execution, we moved from loosely monitored projects to closely monitored projects with well defined feedback loops. In the beginning we collected too much data, independent of the process. Later data became embedded in the process. The types of analysis we performed in the beginning were correlations and regressions, and we have evolved to other forms of model building, based upon the nature of the software engineering data, and to the use of qualitative analysis. Packaging began as recording and generating lessons learned but evolved to focused tailored packages that were integrated into the development processes. We started by packaging defect and resource baselines and product characteristics and have been evolving to seeking the relationship between process and product characteristics.

1976 - 1980

What we did

We began the SEL in 1976. At that time, the paradigm looked like:

1. ~~Characterize/Understand~~ Apply Models
2. ~~Set Goals~~ Measure
3. ~~Select Process~~ Study Process
4. Execute Process
5. Analyze Data Only
6. ~~Package~~ Record

We tried to characterize and understand by using other people's models. For example we spent a great deal of time trying to apply such models as the Rayleigh curve model of resource allocation, reliability growth models, etc. without asking ourselves if they were appropriate for our particular environment.

We decided on measurement as an abstraction mechanism and developed data collection forms and measurement tools. We collected data from half a dozen projects for a simple data base and we defined the GQM as in informal mechanism to help us organize the data around the study of defects [BaWe84].

It had not really occurred to us to select process as we did not yet understand that process was a variable that needed to be selected and tailored to the environment. This was because we had not yet understood our environment sufficiently. So we started to study process, applied heuristically defined combinations of existing processes and began to run controlled experiments at the university with students.

During development, data collection was an add-on activity and was loosely monitored. We analyzed data only and began to build baselines and looked for correlations. We recorded what we found, built defect baselines and resource models and measured project characteristics.

What we Learned

During this period we learned that we needed to better understand the environment, projects, processes, products, etc. We needed to build our own models to understand and characterize our environment, we could not just use other people's models. Those models were built for their environments and could not be generalized easily.

We learned that we needed to understand what factors create similarities and differences among projects so we know the appropriate model to apply. This included the need to understand how to choose the right processes in order to create the desired product characteristics.

We realized that evaluation and feedback are necessary for project control and that data collection has to be goal driven; we could not just collect data and then figure out what to do with it.

From our perspective, the major improvement technology that emerged from this period was the Goal/Question/Metric Paradigm, even though it was still quite primitive.

An Example

As an example of what we learned, we tried to apply the 40/20/40 rule in SEL. It had been reported by Boehm [Bo73] that approximately 40% of project resources were expended in analysis and design, 20% in code, and 40% in checkout and test. Shortly thereafter, Walston and Felix reported that in IBM/FSD, 35% of the resources were expended in analysis and design, 30% in code, 25% in checkout and test and 10% in other, which clearly violated the 40/20/40 rule [WaFe77]. But in the SEL, we were collecting two types of resource data, phase data and activity data. The phase data represented milestone data. That is, analysis and design data represented the resources expended up to the design review milestone (CDR). The activity data represented what a developer did each week, e.g., 20 hours designing, 10 hours coding, 5 hours in training, 5 hours in travel. Using the phase data, we found that 20% of the resources were expended in analysis and design, 45% in code, 28% in checkout and test and 5% in other, while using the activity data, we found that 21% of the resources were expended in analysis and design, 28% in code, 23% in checkout and test and 27% in other.

	TRW	**IBM**	**SEL**	
			Phase	**Activity**
Analysis/Design	40%	35%	20%	21%
Code	20	30	45	28
Checkout/Test	40	25	28	23
Other		10	5	27

Table 1. Resource Allocation Data

It became clear that the data from the other environments represented phase data rather than activity data since they did not collect activity data. It also was clear that each of the organizations defined their milestones and phases differently, so each organization has a different model for resource allocation and it is hard to compare them. Phase data is highly dependent on how an organization defines its milestones. Since phase data and activity data represent two entirely different things, it is not clear what the activity data look like in these other organizations. It should be noted that this example represents an argument why it would be very difficult to build a national data base across environments and share and compare data.

1981 - 1985

What we did

In the early eighties, the paradigm had evolved to look more like:

1. Characterize/Understand
2. Set Goals
3. Select Process
4. Execute Process
5. Analyze
6. ~~Package~~ Record

To characterize and understand the environment we built our own baselines/models of cost, defects, process, etc. We began to set goals for all data collected and expanded our definition of the GQM to perform studies across multiple areas and projects. We began to incorporate subjective metrics into our measurement process. To help us select process we experimented with well defined technologies and began experiments with high impact technology sets, e.g., Ada & OOD. During project execution, we collected less data than we had before and moved the data from a file system to a commercial, relational data base. We began to understand how to combine some of our off-line controlled experiments with the case studies in the SEL. We shifted the analysis emphasis to the process and its relation to product characteristics. We recorded lessons learned, and began formalizing processes, products, knowledge and quality models.

What we Learned

During this period we learned that software development follows an experimental paradigm, i.e., you need to set your goals up front and check that you are achieving those goals. The design of experiments is an important part of improvement and evaluation and feedback are necessary for learning. We also learned that we needed to better understand relationships between various kinds of experiences, e.g., the relationship between processes and the set of product characteristics it evokes or the resources required to perform it, the relationship between component size and complexity and defect rate. To do this process, product, and quality models need to be better defined, experimentally tested, and improved.

We learned that reusing experience in the form of processes, products, and other forms of knowledge is essential for improvement. We need to learn what works and what does not work and what needs to be modified and what needs to be thrown out. At the same time we need to experiment with new technologies, motivated by our experiences.

By this time, we had more data than we knew what to do with them, but we did not have the data we needed to help us interpret what was happening. We learned that you can drown in too much data, especially if you don't have goals. Besides having a good data base, you need to store your models as well as your data .

An Example

As an example of demonstrating that we need to understand the relationship between variables, consider the study in the SEL where we compared fault rate with component size and complexity. In a study in the early eighties, we found that the simple minded view that defect rate increases with size did not hold in the SEL environment. In fact, we found the opposite for the actual data we had available for study [BaPe84]. We believe this relationship is due to the fact that interface defects dominate the problem of the complexity of the individual component, when components are small.

On the other hand, we have hypothesized that as the size grows beyond the developer's ability to cope with its size and complexity, the complexity of the individual component will dominate the complexity of the interface and fault rate will again grow.

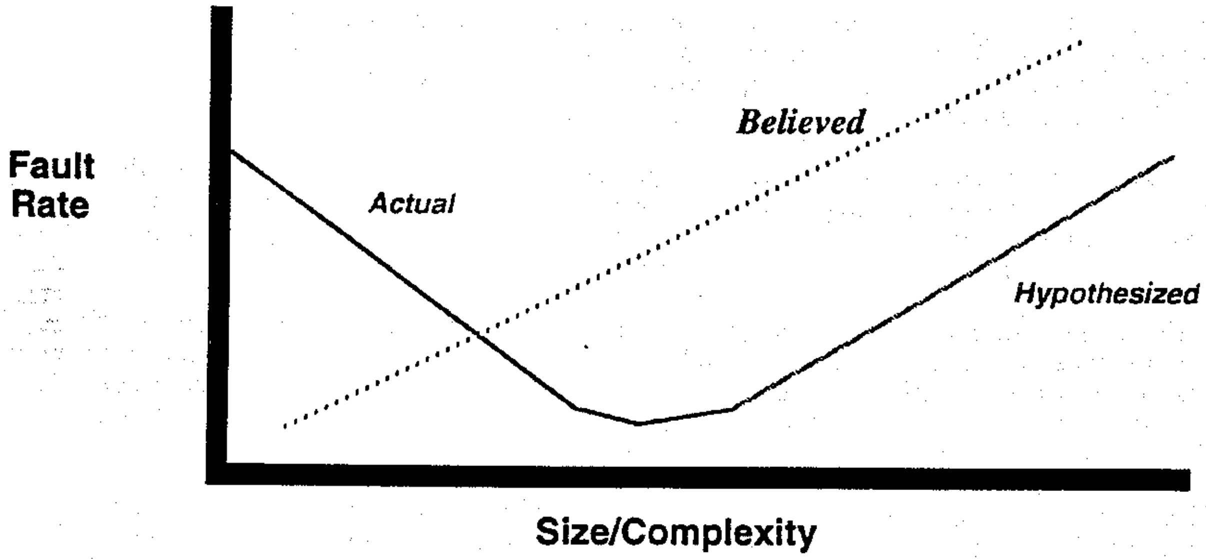

Figure 1. Relationship between Fault Rate and Size or Complexity

We have since found support for the first statement, i.e., fault rate decrease with size and complexity in data from several companies. This result was a surprise at the time since most people believed that smaller components were better. However. the relationship between size and fault rate appears not to be that simple.

1986 - 1990

What we did

It was in this period that the QIP took its current form, recording being changed to packaging.
1. Characterize/Understand

2. Set Goals
3. Select/Tailor Process
4. Execute Process
5. Analyze
6. Package

To characterize and understand we worked on capturing experience through models. Goals and models became the commonplace driver of measurement and we built SME [Va87], a model-based experience base with dozens of projects. We began to tailor and evolve high impact technologies based on experience, e.g., Cleanroom, and experimentation and feedback became an integral part of the QIP. During process execution, we embedded the data collection process into the development processes and more closely monitored projects, especially those where we were experimenting with new approaches.We began to demonstrate various (process, product) relationships, e.g., the effect of a particular method on defect reduction. We developed focused tailored packages, e.g., generic code components, and learned to transfer technology better through organizational structure, experimentation, and evolutionary culture change.

What we Learned

We learned that experience needs to be evaluated, tailored, and packaged for reuse. That is, you just cannot write lessons learned documents, you have to analyze and synthesize what has been learned and integrate it into the existing knowledge so that it is usable by future projects. This requires organizational support and resources.

A variety of experiences can be reused, e.g., process, product, resource, defect and quality models. But processes must be put in place to support the reuse of experience and the development process must be modified to take advantage of reusable experiences. Experiences can be packaged in a variety of ways, e.g., equations, histograms, algorithms.

Packaged experiences need to be integrated. When introducing a new process, an organization needs to make sure it fits and is supported by the other processes being used, that is, it needs to understand the relationship between various changes in the parameters in one model and the effect on another model. If I modify my reading technology, what will be the effect on the class of defects I find, the resources allocated for rework, etc.

There is a tradeoff between reuse and improvement. Evolution is slow as I cannot introduce too much change at one time. When I do introduce change, I loose experience and predictability. On the other hand, processes have to be changed to cope with the continuously growing need for quality.

During this period we evolved the GQM to include templates and models [BaRo88] and formalized the organization via the Experience Factory Organization [Ba89].

An Example

To demonstrate that how a technology is packaged and integrated has a strong effect on its effectiveness, consider our experiences with evaluating and integrating reading technology. We ran a controlled experiment comparing equivalence partitioning testing, structural testing, and reading by step-wise abstraction[BaSe87]. Reading was found to be more effective and efficient than testing in uncovering defects. Based upon these results, we put reading into practice as a technology in the SEL. But we found that reading had little effect on defects. This appeared to be because the readers did not read well because they knew they were going to test and believed that, in spite of the experimental results, testing was better. Our belief that reading is more effective when not followed by developer testing motivated our use of the Cleanroom approach [SeBaBa87]. When embedded in the Cleanroom approach, reading did demonstrate a substantial lowering of defect rates.

1991 - 1995

What we are doing

This bring us up to the current time. The current evolution of the QIP appears to be aimed at instantiating the steps, making them more specific, providing details, and developing support technologies.

To characterize and understand the project and environment, we are building a repository of (process,product) relationship models that characterize the SEL environment. We are working on automating the GQM in order to support the setting of goals. We are studying what experience is exportable to other environments in help other organizations take advantage of our process experience We are working on building models to measure process conformance and domain understanding.

During execution of the processes, we are working to capture the details of experience by providing more interaction between developers and experimenters and more effective feedback mechanism. This will help us to evolve processes that are more focused and detailed for our local needs and goals.

We are building qualitative analysis approaches to extract our experiences and provide input to the data models. We continue to evolve SME and we continue the evolution and packaging of the Experience Factory Organization.

Many of the current, specific SEL activities are covered in this workshop proceedings. However, there are more global SEL activities aimed at evolving the application of the QIP to other organizations. These activities concern packaging the SEL organizational experience for other groups in NASA, understanding whether and how to move activities to common use, and better integrating reuse into the development process

The research activities are based upon instantiating the steps of the Quality Improvement Paradigm by providing support technologies and automation, and integrating the various activities.

Where the research is going

The table below shows some of our current research interests aimed at instantiating the Quality Improvement Paradigm.

Step	Studies / Research Projects
Characterize	Perform domain analysis to identify similar projects using techniques appropriate for SE data
Set goals	Automate the model-based GQM as much as possible

Choose process	Develop technologies tailorable to the specific project needs
Execute processes	Build a more powerful, flexible experience base
Analyze data	Learn how to run more efficient experiments and combine controlled experiments with case studies
Package experience	Build better models and modeling notations

Table 2: Instantiating the Quality Improvement Paradigm

Example research projects

To give some specific examples of research projects, let us consider three: the work on domain analysis, reading technologies, and empirical modeling.

Domain Analysis

Problem Addressed:
How do you recognize which projects are most like yours in order to use the experiences from these projects to allow you to build models, choose similar process, etc.?

Current Status:
We have established procedures to identify and analyze software domains within and across organizations so that opportunities for reuse of experiences may be identified [Lionel Briand]. This has entailed defining both an experience-based procedure taking advantage of intuition and expert knowledge as well as a data-based procedure for when data is available.

Validation Strategy:
We are using both procedures to identify domains within NASA, and have analyzed data within the SEL data base to determine whether or not our assumptions are supported locally.

Focused Tailored Reading Techniques

Problem Addressed:
How do you tailor a process to the project goals and local organizational characteristics?

Current Status:
Have developed scenario-based technologies for reading various documents that are tailorable and can be focused for the particular environment. As an example, we have developed several model-based scenarios that take advantage of local knowledge and technical models to define a technology for reading. For example, defect-based reading is based upon the different defect classes, e.g., missing functionality, data type inconsistencies, in a requirements document that have been found in requirements [BaWe81].

Validation:
We have run a couple of controlled experiments that show that defect-Based reading is significantly more effective that ad hoc reading or checklists [PoVo94].

Empirical Modeling: Optimized Set Reduction

Problem Addressed:
How do you build empirical models that allow you to define interpretable, accurate, easy to use and automate modeling procedures that take into account the specific constraints of software engineering data?

Current Status:
OSR has been developed based on pattern matching; searching for similar experiences in the data set and the use of non-parametric statistics. There are no functional assumptions made; the approach handles interactions and inter dependencies among variables, and no "learning" parameters need to be tuned before hand.

Validation:
We have shown OSR to be easier to interpret and more accurate than regression and tree-based approaches for cost modeling and defective module prediction [BrBaTh92, BrBaHe93]. A prototype tool exists and a commercial tool is under development.

Conclusion

Over the past 18 years we have learned a great deal about software improvement. Our learning process has been continuous and evolutionary like the evolution of the software development process itself. We have packaged what we have learned into our process, product and organizational structure. This evolution is supported by the symbiotic relationship between research and practice. It is based upon a belief that software engineering is a laboratory science. As such it involves the interaction of research and application, experimentation and development. It is a relationship that requires patience and understanding on both sides, but when nurtured, really pays dividends!

References

[Ba85]
V. R. Basili, "Quantitative Evaluation of Software Engineering Methodology," Proc. of the First Pan Pacific Computer Conference, Melbourne, Australia, September 1985 [also available as Technical Report, TR-1519, Dept. of Computer Science, University of Maryland, College Park, July 1985].

[Ba89]
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

[Ba90]
V. R. Basili, "Software Modeling and Measurement: The Goal/Question/Metric Paradigm," University of Maryland Technical Report, CS-TR-2956, UMIACS-TR-92-96, September 1992.

[BaRo88]
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.

[BaPe84]
V. R. Basili, B. Perricone, "Software Errors and Complexity: An Empirical Investigation," ACM Communications, vol. 27, no. 1, January 1984, pp. 45-52.

[BaSe87]
Victor R. Basili, R. W. Selby, "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering, Vol. SE-13, No. 12, December 1987, pp. 1278-1296.

[BaWe84]
V. R. Basili, D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, vol. SE-10, no.6, November 1984, pp. 728-738.

[BaWe81]
V. R. Basili, D. M. Weiss, "Evaluation of a Software Requirements Document by Analysis of Change Data," Proceedings of the Fifth International Conference on Software Engineering, San Diego, USA, March 1981, pp. 314-323.

[Bo73]
B. W. Boehm, "Software and its Impact: A Quantitative Assessment," Datamation 19, No.5 48-59 (My 1973).

[BrBaTh92]
Lionel C. Briand, Victor R. Basili, and William M. Thomas, "A Pattern Recognition Approach for Software Engineering Data Analysis," IEEE Transactions of Software Engineering, Vol. 18, No. 11, pp. 931-942, November 1992.

[BrBaHe93]
Lionel C. Briand, Victor R. Basili, and Christopher J. Hetmanski, "Developing Interpretable Models for Identifying High Risk Software Components," IEEE Transactions on Software Engineering, November 1993.

[PoVo94]
Adam Porter, Larry Votta, "An Experiment to Assess different Defect Methods for Software Requirements Inspections," Proceedings of the 16th ICSE, Sorrento, Italy, May 1994.

[SeBaBa87]
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.

[Va87]
J. D. Valett, "The Dynamic Management Information Tool (DYNAMITE):Analysis of the Prototype, Requirements and Operational Scenarios," M.Sc. Thesis, University of Maryland, 1987.

[WaFe77]
C. E. Walston and C. P. Felix, "A Method of Programming Measurement and Estimation," IBM Systems Journal, Vol. 16, No. 1, 1977, pp.54-73.