

The Experience Factory: Packaging Software Experience

Since software deals with man-made artifacts, we need to view software development as an experimental science and build models of the artifacts and the processes by which they are manufactured. To do this we need to isolate and categorize the components of the discipline, define notations for representing these components, and specify the interrelationships among these components as they are manipulated. The components of the discipline consist of various processes (e.g., life cycle models, methods, techniques, tools), products (e.g., code components, requirements, designs, specifications, test plans), and other forms of experience (e.g., resource models, defect models, quality models, economic models).

We need to build descriptive models of the discipline components to better understand (1) the nature of the processes and products and their various characteristics, (2) the variations among them, (3) the weaknesses and strengths of both, and (4) mechanisms to predict and control them. Models exist for some components, for example there are several mathematical models of programs and modules, there are parametrized cost models that attempt to predict the cost of a project based upon past experience, there are informal descriptions of the life cycle process. However many more models are needed and those models that do exist need to be more formally defined and further analyzed and integrated to provide a deeper understanding of the components and their interactions.

Based upon analysis of these descriptive models, we need to build prescriptive models that improve both products and the processes for creating them relative to a variety of qualities, provide feedback for project control, and allow the packaging of successful experience. Because the overall solutions are both technical and managerial, model building requires the support of a variety of disciplines both within and outside the discipline.

The overall approach requires an approach, similar to the scientific method, that allows us to experiment, measure, learn, build better models, and reuse past experiences. For the past 15 years, we have been applying such an approach, the Quality Improvement Paradigm, in the Software Engineering Laboratory (SEL) at NASA/GSFC. It adapts the scientific method to software development. As stated in the first chapter, the basic steps involve:

Planning: an iterative process involving characterizing the current project and its environment, setting the quantifiable goals for successful project performance and improvement, and choosing the appropriate process model and supporting methods and tools for this project.

Execution: a closed-loop project cycle which involves executing the processes, constructing the products, collecting and validating the prescribed data, and analyzing it in real-time to provide feedback for corrective action on the current project.

Analysis and Packaging: a post mortem analysis of the data and information gathered to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements, and a packaging of the experience gained in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects and the storing of the packages in an experience base so it is available for future projects.

The Software Engineering Laboratory Experience

In the evolution of the SEL, we can identify three three major phases, understanding, assessing and improving and packaging. During the first phase, we worked on understanding the environment and how to measure it. To achieve this, we measured what we could, e.g., resources and defects, applied whatever models existed, e.g., cost models and reliability models, built baselines for such things as defect classes and resource allocation, and developed the Goal/Question/Metric paradigm as an organized mechanism for setting goals and measuring the software project.

During this phase we learned that although there are similarities among project environments, the differences require that each project be treated differently, i.e., different processes, methods and techniques need to be tailored to the specific needs of the project. Thus there is not one standard life cycle process for all software projects. We learned that there is a direct relationship between the processes performed and the various product qualities. Thus, one can effect specific product qualities by making specific changes in the process. We learned that measurement needs to be based upon goals and models. Performing measurement bottom-up neither provides the right metrics nor the means for interpreting them. We learned that evaluation and feedback are necessary for project control, i.e., providing managers with the right data helps them manage the project in a more effective way.

In phase two, we worked to improve the process and product quantitatively based upon the evolutionary development of various models. To this end, we experimented with various reading and testing technologies (reading by stepwise refinement, functional testing, structural testing), various design technologies (function decomposition, object oriented design), various life cycle models (waterfall, Cleanroom) by running controlled experiments and case studies to determine the effectiveness of each of these technologies in the environment. Based upon the proposed models, we evaluated and feedback information to the project managers. This allowed them to better estimate the resources required, predict the kinds of defects that take place in their environment, and understand which aspects of their development process that were at risk. We formally developed the Quality Improvement Paradigm and began formalizing process, product, and quality models for the environment. This allowed us to tailor what we learned about the effectiveness of reading and testing for the SEL environment and change the basic processes based upon the results of prior projects. Naturally the GQM and various other models continued to evolve.

During this phase we better understood the experimental nature of software development and how to apply the Quality Improvement Paradigm to the resulting processes. This entailed defining and refining process, product, knowledge and quality models. The standard SEL FORTRAN development model

was more precisely defined. For example, changes were made to take advantage of the benefits of reading technologies. Variations of the model were created as a result of tailoring it for different project classes that occurred in the SEL. In this way we were able to evaluate and feedback what we had learned from prior projects in the environment to improve the environment for future projects.. We learned that our prior experiences needed to be packaged in a way that allows the incorporation of new knowledge and tailoring to the characteristics of the current project. In this way we could reuse experience in the form of processes, products, and other forms of knowledge to improve software.

We are now in the third phase, whose goal is to package all kinds of experience for reuse. This phase involves choosing the appropriate experiences with the highest potential for reuse, studying and experimenting with notations for defining reusable experiences, defining models of tailorable, reusable experiences, and defining process models for development and maintenance that support reusing experiences.

So far we have learned that a variety of experiences can be packaged and in a variety of ways. We have also learned that these packaged experiences need to be integrated into a form that allows a better understanding of their interrelationships, for example, we need to understand how a change in one activity within a particular process model will effect the other activities, the resource consumption, the reliability of the product, etc.

In the next several sections we will review the steps of this paradigm and present more detail on those steps that have been omitted.

Characterizing the Environment

Characterizing the environment requires that we classify the current project with respect to a variety of characteristics. It allows us to isolate the the class of projects with similar characteristics and goals to the project being developed. This way we can distinguish the relevant project environment for the current project. Characterization provides a context for goal definition, reuse of experience and objects, process selection, evaluation and comparison, and prediction.

There are a large number of project and environmental characteristics that affect the software development process and product [Ba81a]. These include *people factors*, such as the number of people, level of expertise, group organization, problem experience, process experience; *problem factors*, such as the application domain, newness to state of the art, susceptibility to change, problem constraints; *process factors*, such as life cycle models, methods, techniques, tools, programming language, other notations; *product factors*, such as deliverables, system size, required qualities, e.g., reliability, portability; and *resource factors*, such as target and development machines, calendar time, budget, existing software.

Setting the Goals: The Goal/Question/Metric Paradigm

We need to establish models and goals for the processes and products. These goals should be measurable, driven by models. They can be defined from a variety of perspectives, e.g., the user, customer, project, corporation. There are a variety of mechanisms for defining measurable goals: the Quality Function Deployment Approach (QFD), the Goal/Question/Metric Paradigm (GQM) [BaWe84, BaSe84, BaRo88, Ba90b], and the Software Quality Metrics Approach (SQM) [BoBrLi76].

The Goal/Question/Metric Paradigm is the mechanism used in the planning phase of the Quality Improvement Paradigm for defining and evaluating a set of operational goals, using measurement. It represents a systematic approach for tailoring and integrating goals with models of the software processes, products and quality perspectives of interest, based upon the specific needs of the project and the organization.

The goals are defined in an operational, tractable way by refining them into a set of quantifiable questions that are used to extract the appropriate information from the models. The questions and models, in turn, define a specific set of metrics and data for collection and provide a framework for interpretation of the data.

The Goal/Question/Metric paradigm combines models of an **object of study**, e.g., a process, product, or any other experience model and one or more **focuses**, e.g., models aimed at viewing the object of study for particular characteristics that can be 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/information is made available for any **purpose**, e.g., characterization, evaluation, prediction, motivation, improvement, which specifies the type of analysis necessary to generate a **GQM model** relative to a particular environment.

To aid in developing goals, there is a goal generation template. 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. The goal is defined based upon filling in a set of values for the various parameters in the template. Template parameters include purpose (what object and why), perspective (what aspect and who) and the environmental characteristics (context).

Several examples of goal development were given throughout this text.

Choosing the Execution Model

Choosing the execution model involves choosing and tailoring an appropriate generic life cycle model, set of methods, and techniques. It should be noted that choosing and tailoring any form of process involves providing its goal and procedure definition. Choosing and tailoring are always performed in the context of the environment, project characteristics, and goals established for the products and other processes.

In order to aid in the selection of the appropriate processes, we need to establish goals for the software development process from a variety of perspectives. These goals need to be articulated and evaluated.

Executing the Processes

In order to execute the processes, we need access to packaged experience in the form of the processes we have chosen, prior products available for reuse, the appropriate resource and data models, and a software development model that allows us to take advantage of reusable packages. We need to analyze the data according to the project specific models and goals in close to real time in order to make the project visible to management and feed back information for corrective action. Data collection must be integrated into the processes, not considered as an add on, e.g. the defect classification form should be part of the configuration control mechanism. Data validation is important to assure we are making decisions based upon valid data. It is clear that automation is necessary to support the mechanical tasks, deal with the large amounts of data and information, and aid in the data analysis.

There is a wide variety of data that can be collected. resource data includes effort by activity, phase, type of personnel, computer time, and calendar time. Change and defect data include changes and defects by various classification schemes. Process measurement includes process definition, process conformance, and domain understanding data. Product data includes product characteristics, both logical,(e.g., application domain, function) and physical,(e.g. size, structure) and use and context information.

Analyzing and Packaging Experience: The Experience Factory

The Quality Improvement Paradigm is based upon the notion that improving the software process and product 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*). The paradigm implies the separation of project development (performed by the Project Organization) from the systematic learning and packaging of reusable experiences (performed by the Experience Factory).[Basili 89]

The Experience Factory is a logical and/or physical organization that supports project developments by analyzing and synthesizing all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand. 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.

The analysis and interpretation of the data collected is based upon the goals. We can use this data to: characterize and understand, (e.g., what project characteristics effect the choice of processes, methods and techniques?, which phase is typically the greatest source of errors?); evaluate and analyze, (e.g. what is the statement coverage of the acceptance test plan? does the Cleanroom Process reduce the rework effort?); predict and control, (e.g., given a set of project characteristics, what is the expected

cost and reliability, based upon our history); and motivate and improve, (e.g., for what classes of errors is a particular technique most effective).

Systematic learning requires support for recording experience, off-line generalizing and tailoring of experience, and formalizing experience. Packaging useful experience requires a variety of models and 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. Systematic reuse requires support for using existing experience, and on-line generalizing or tailoring of candidate experience. This combination of ingredients requires an organizational structure that supports them. This includes: a software evolution model that supports reuse, a set of processes for learning, packaging, and storing experience, and the integration of these two functions.

It is important to understand that the term reuse is used here to mean more than product reuse. But it includes product reuse. Reuse, in the software domain, has been a long sought after goal with little historical success. Why has reuse been a problem in the software domain? There are several reasons. First we need to reuse more than just code, we need to reuse context. Second, the reuse of experience has been too informal, it has not been fully incorporated into the development or maintenance process models. Third, we have not done an effective job of analyzing and evaluating experience for its reuse potential, nor have we packaged it appropriately. Fourth, we have assumed that reuse means reuse as is, most experiences need to be tailored in some way to meet the needs of the new context. Lastly, it was assumed that reusable packages of experience, be it product, process or any other form of experience, could be developed as a by-product of the project. Clearly this is not the case. The project focus is delivery, not reuse. If we want reuse, the activities that create reusable packages of experience need to be outside of the project. This is why we need separate organizations, the Project Organization for product development and the Experience Factory for packaging experience. Both organizations have different focuses and priorities, use different process models, and have different expertise requirements. Trying to mix them in the same organization is destined to failure [BaCa88, BaRo89, BaRoBaJo87].

Examples of Packaged Experience

We can package all kinds of experience. We can build resource models and baselines, change and defect models and baselines, product models and baselines, process definitions and models, method and technique models and evaluations, products and product models, a library of lessons learned, and a variety of quality models.

There are a variety of forms for package experience. There are equations defining the relationship between variables, (e.g. $\text{Effort} = a * \text{Size}^b$), histograms or pie charts of raw or analyzed data, (e.g. % of each class of fault), graphs defining ranges of normal (e.g., graphs of size growth over time with confidence levels), specific lessons learned associated with project types, phases, 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), and 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).

Examples of resource models and baselines include cost models, resource allocation models for staffing, schedule, and computer utilization, and the relationship between resources and various factors that effect resources, e.g. specific methods, customer complexity, the application, the environment, and defect classes. [BaBa81, BaBe81, BaFr81, BaPa85, BaZe78, Bo81, JeBa88]

In building resources models, we are interested in capturing data associated with a variety of factors associated with prior projects, e.g., size, effort, pages of documentation, calendar time. We can use these relationships to build equations that define the empirical relationships between these factors. Using this data we can either generate separate equations representing characteristically different environments (based upon characterizing factors), or we can use the characterizing factors to adjust the equations to provide better fits to the range of data. These relationships package the organization's experience with respect to various resources. The characterizing factors also provide insight into those factors that effect resources. The equations can be used for prediction, project monitoring, and evaluation.

In the SEL, examples of packaged relationships include:

$$\text{Effort} = 4.37 + 1.43\text{devlines}$$

$$\text{Effort} = 5.5 + 1.5\text{newlines}$$

$$\text{Docpages} = 99.1 + 30.9 \text{ devlines}$$

$$\text{Numruns} = -108 + 151\text{devlines}$$

Projects under 50kloc:

$$\text{Effort} = .877 + 1.5\text{newlines}$$

Projects over 50kloc

$$\text{Effort} = 66.9 + .003 \text{ numruns}$$

We have also been able to demonstrate that various methodology factors favorable impact cost and quality and various complexity factors unfavorable impact cost and quality

Examples of change and defect models and baselines include: change baselines by various classifications, defect baselines by various classifications, defect prediction models, reliability models. [WeBa85, BaPe84, BaRo87, BaWe81, SeBa88]

We can capture the number of errors, faults and failures associated with various phases, e.g., in total, by various classes (error domain, time of detection, omission/commission, software aspect, failure by resolution date opened/date closed). We can build histograms and cumulative graphs as baselines for the various defect classes and identify overall patterns as well as common patterns representing characteristically different environments. These defect distributions package the organization's

experience with respect to defects associated with various project characteristics. They can be used for prediction, project monitoring, evaluation, and provide specific focuses for improvement.

Examples of project characteristic models and baselines include; growth and change histories for size, staffing, computer use, number of test cases, test coverage, etc. These can be compared with the norm for the environment or for a set of characteristically similar projects to make predictions and estimates for the current project and provide guidance for the project manager based upon variation from expectation [Mc85, RaBa85]

For a variety of data we can define graphs of various variables over time. The accumulation of such data over a variety of projects provide baselines for planning and monitoring projects. For example, we can plot: growth in lines of code vs. schedule, CPU hours vs. calendar time, the amount of code covered vs. number of tests run, the amount of reuse in each project over time.

We can package our experiences with techniques, methods, and life cycle models by defining and refining models of their definitions and goals, understanding where they are appropriate and how they need to be tailored to a particular set of environmental characteristics. To focus on improvement, we need to introduce new technology. We need to experiment and record the findings in terms of lessons learned and eventually adjustments to the current processes. When the technology is substantially different from what we are currently using, the experimentation may be off-line. It may take the form of a controlled experiment (for detailed evaluation in the small) or as part of a case study (to study the effects of scale up). In both cases, the goal/question/metric paradigm provides an important framework.[BaSeHu86]

Based upon experimentation, we can write a set of lessons learned that can be made available in the data base for future uses of the technology. We can define new methods and techniques or refine old ones.

One such experiment we performed in the SEL was the comparison and evaluation of various testing and reading techniques discussed in an earlier chapter. [BaSe87] Lessons learned from this experiment include: code reading by stepwise abstraction is more effective and cheaper in human and computer costs than functional or structural testing, code readers have a better assessment of the quality of the product, code reading is better for interface and computation faults, while functional testing better for control flow faults, and structural testing is weak for omission faults.

We have used the QIP for developing new or refined process models. To do this one uses the project characteristics and goals to find the most appropriate process models for the current project. One then develops, modifies or refines the process based upon the lessons learned from previous applications of the model. Goals are set to monitor those new or high risk areas. The model is executed, data collected and analyzed to make changes to the process in real time. Based upon the goals and analysis, one then records lessons learned and modifies the process for future use.

When changing to an Ada life cycle model for software development in the SEL, we first examined the existing models in the experience base. These included the standard SEL FORTRAN models and an experimental Ada model generated while running and experiment at GE for a similar application [BaKaPaLoCh85]. Goals were defined to characterize and evaluate in general, and with respect to reuse. Lessons learned for the GE/Ada model include: 1 month training was insufficient, i.e., more training and experience were needed in OOD; the requirements need rewriting (they were too low level and FORTRAN oriented), and finding a production quality compiler and environment is a potential problem.

Based upon these lessons learned, we defined an SEL/Ada process model. Initial steps included a evaluations phase for choosing the appropriate Ada compiler and environment (we chose DEC Ada), 3 months training in Ada and OOD, a cost model that recognized the lose of prior requirements analysis, design and code experience, a phase for analyze existing requirements and rewriting them to be more compatible with an Ada development, etc. When no information was available from the experience base, we used the standard SEL activities.

Then, we monitored the project based upon the goals established and made changes to the process model in real time when needed. We wrote lessons learned at the end of each phase at at the end of the project for incorporation into next version. The redefined process was then available for the next execution of of an Ada project. Since then it has been continuously modified and tailored. [Ag86, BrAgBa87]

Some lessons learned in the use of object-oriented design include: the requirements language (composite specification models) succeeded in removing the FORTRAN biases from the original specifications, tailoring an OOD methodology to the SEL environment was essential, graphical and textual representations of the design were both useful, training is needed in OOD for everyone, including high-level management.

Some lessons learned in the use of Ada include: the use of library units made the library structure more complex, however, deeply nested structures had several more serious disadvantages, (e.g., it increased compilation costs, made reading more difficult, made reuse harder, made it harder to locate particular procedures, made unit test harder); task use should be minimized in the SEL environment because they are minimally necessary, inherently difficult, easily proliferate and make reading by less experienced people difficult; type analysis is necessary during early design (there can be a proliferation of types because of strong typing) so use of subtypes essential and types should be placed at the lowest level of design possible; unit testing in Ada needs careful definition because deep nesting affects units, module-to-module coupling in Ada is higher, and adding write statements causes recompilation.

Another process model tailored for the SEL environment was the Cleanroom model proposed by Mills. Here the relevant existing models in the experience base include: the standard SEL models, the IBM/FSD Cleanroom Model [Dy82], and a Cleanroom model used for a controlled experiment at the University of Maryland [SeBaBa87].

Again, goals were defined to characterize and evaluate in general, and specifically with respect to changing requirements. Lessons learned from the IBM/Cleanroom model included: the basic process model, methods and techniques, results that the process was very effective in given environment. Lessons learned from the University of Maryland/Cleanroom model include: no testing by developers enforces better reading, the process is quite effective for small well specified projects, formal methods are hard to apply, they require skill development, and there may be insufficient data to measure reliability appropriately.

We defined the SEL/Cleanroom process model to use informal state machines and functions for programs and modules respectively; training was consistent with University of Maryland course on Cleanroom process model, methods, and techniques; the emphasis was on reading by two reviewers; back-out options were allowed for unit testing certain modules because of the concern for reading certain types of equations. Again, when no other information was available in the experience base, we used standard SEL activities.

Again, we monitored the project based upon the goals established, especially with regard to risk and made changes to the process model in real time when needed. We recorded lessons learned at the end of each phase at the end of the project for incorporation into next version, assuming the project was successful.

Some lessons learned in the application of the Cleanroom process model include: we can scale up to 30KLOC, can use the approach in an environment with changing requirements, the failure rate during test was reduced to almost 50% of the normal failure rate, there was a reduction in rework effort, (95% as opposed to 58% took < 1 hour to fix), only 26% of faults found by both readers, productivity increased by about 30%, the effort distribution data showed more time in design than the standard project and 50% of code time was spent reading, code appears in the library later than normal and more like a step function, there was less computer use by a factor of about 5, better training is needed for the methods and techniques, better mechanisms are needed for transferring code to testers, testers need to add requirements to the project for analysis of code output, there is no payoff in reliability modeling because of the lack of sufficient data on a project of this size.

One side effect of this project was that it caused more emphasis on requirements analysis. The next experiments were defined with goals to apply the formal models more effectively (using the box structure approach), change the application domain and keep size the same in one project and scale up to a 100KLOC in another project.

Product experience that can be packaged includes appropriately parametrized code components, designs, specifications, requirements, and test plans [BaCa88]. Several studies are underway with regard to product reuse, especially for the Ada environment.

Automation of the experience models and baselines is very important. Automated support has been developed in the form of a measurement environment, Software Management Environment, SME, at NASA/GSFC that supports the data, models and

provides management with visual support for analysis, assessment, prediction, and guidance during the project, and records new project data in the data base [Va87, DoBa85, LoBa89].

Experience Factory Implications

The Experience Factory offers an organizational structure that separates the product development focus from the learning and reuse focus. It supports learning and reuse and generates a tangible corporate asset in the form of packaged experiences. It aids in the formalization of management and development processes. It links focused research with development.

The Experience Factory can be used to consolidate and integrate activities, such as packaging experience, consulting, quality assurance, education and training, process and tool support, and measurement and evaluation.

The Experience Factory makes existing technologies more relevant, e.g., verification techniques for product packaging. It forces research to focus on corporate needs and technology transfer. Areas of research for supporting the activities include defining and tailoring models, the integration of technologies, scaling-up techniques and methods, building and accessing the experience base, and automation.

It makes existing education in formalisms, models and notations more relevant. It requires education in verification technologies, formal requirements and specification notations, formal models of measurement and management, and assessment technologies.

How the experience factory is funded depends upon the organizational structure of the corporation. Clearly the project organization and the experience factory should be separate cost centers, initially funded out of corporate overhead. However, eventually one would like to have projects billed for packages, so that the factory can be self supporting and focused toward project support.

There are costs involved in instituting such a program. The level of funding clearly depends upon the size of the program. However, some relative data is available. Based upon the SEL experience where a full measurement program has been in progress for over 14 years, project data collection overhead is estimated to be about 5% of the total project cost. Although our experience shows that this typically does not effect total project cost, since the data collection activity pays for itself on first project in terms of improvement, it must be established as an up-front cost. With regard to the Experience Factory, the costs depend upon the number of projects supported, level of effort and set of activities performed, e.g., quality assurance, process definition, tool building, education and training, etc. One might consider that it takes a minimum of two people, however to create the critical mass necessary to develop such and activity at the minimal level.

Conclusions

The integration of the Quality Improvement Paradigm, the Goal/Question/Metric Paradigm, and the Experience Factory Organization provides a framework for software engineering development, maintenance, and research. It takes advantage of the experimental nature of software engineering.

Based upon our experiences, it helps us understand how software is built and where the problems are, define and formalize effective models of process and product, evaluate the process and the product in the right context, predict and control process and product, qualities, package and reuse successful experiences, and feed back experience to current and future projects. A real strength of the approach is that it can be applied today and evolve with technology. It can start small and expand with the increase of knowledge and experience.

References

[Ag86]

W. Agresti, "SEL Ada Experiment: Status and Design Experience," Proceedings of the Eleventh Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, December 1986.

[BaBa81]

J. Bailey, V. R. Basili, "A Meta-Model for Software Development Resource Expenditures," Proceedings of the Fifth International Conference on Software Engineering, San Diego, USA, March 1981, pp. 107-116.

[Ba81a]

V. R. Basili, "Data Collection, Validation, and Analysis," in Tutorial on Models and Metrics for Software Management and Engineering, IEEE Catalog No. EHO-167-7, 1981, pp. 310-313.

[Ba85a]

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].

[Ba85b]

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.

[Ba81b]

V. R. Basili, "Evaluating Software Characteristics: Assessment of Software Measures in the Software Engineering Laboratory," Proceedings of the Sixth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, 1981.

[Ba89]

V. R. Basili, "Software Development: A Paradigm for the Future," COMSAC, Orlando Florida, September 1989.

[Ba90a]

Victor R. Basili, "Viewing Maintenance as Reuse-Oriented Software Development," IEEE Software, Jan. 1990, pp 19-15.

[BaBe81]

V. R. Basili, J. Beane, "Can the Parr Curve help with the Manpower Distribution and Resource Estimation Problems," Journal of Systems and Software, vol. 2, no. 1, 1981, pp. 47 - 57.

[BACA88]

V. R. Basili, G. Caldiera, "Reusing Existing Software," Technical Report-xxxx, Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland, 1988.

[BAFR81]

V. R. Basili, K. Freburger, "Programming Measurement and Estimation in the Software Engineering Laboratory," Journal of Systems and Software, vol. 2, no. 1, 1981, pp. 47-57.

[BaKaPaLoCh85]

- V. R. Basili, E. E. Katz, N. M. Panlilio-Yap, C. Loggia Ramsey, S. Chang, "Characterization of an Ada Software Development," IEEE Computer Magazine, September 1985, pp. 53-65.
[BaPa85]
- V. R. Basili, N. M. Panlilio-Yap, "Finding Relationships Between Effort and Other Variables in the SEL," IEEE COMPSAC, October 1985.
[BaPe84]
- V. R. Basili, B. Perricone, "Software Errors and Complexity: An Empirical Investigation," ACM Communications, vol. 27, no. 1, January 1984, pp. 45-52.
[BaRo87]
- V. R. Basili, H. D. Rombach, "Tailoring the Software Process to Project Goals and Environments," Proc. of the Ninth International Conference on Software Engineering, Monterey, CA, March 30 - April 2, 1987, pp. 345-357.
[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.
[BaRo89]
- V. R. Basili, H. D. Rombach, "Software Reuse: A Comprehensive Framework," CS-TR-2158, Department of Computer Science, University of Maryland, College Park, Maryland., 1989.
[BaRoBaJo[87]
- V. R. Basili, H. D. Rombach, J. Bailey, and B. G. Joo, "Software Reuse: A Framework," Proc. of the Tenth Minnowbrook Workshop on Software Reuse, Blue Mountain Lake, New York, July 1987.
[BaSe84]
- V. R. Basili, R. W. Selby, Jr., "Data Collection and Analysis in Software Research and Management," Proc. of the American Statistical Association and Biomeasure Society Joint Statistical Meetings, Philadelphia, PA, August 13-16, 1984.
[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.
[BaSeHu86]
- V. R. Basili, R. W. Selby, D. H. Hutchens, "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, vol. SE-12, no. 7, July 1986, pp. 733-743.
[BaTu75]
- V. R. Basili, A. J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," IEEE Transactions on Software Engineering, vol. SE-1, no. 4, December 1975.
[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.

[BaZe78]

V. R. Basili, M. V. Zelkowitz, "Analyzing Medium Scale Software Development," Proceedings of the Third International Conference on Software Engineering, Atlanta, Georgia, USA, May 1978, pp. 116-123.

[Bo81]

B. W. Boehm, "Software Engineering Economics," Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Bo86]

B. W. Boehm, "A Spiral Model of Software Development and Enhancement," ACM Software Engineering Notes, vol. 11, no. 4, August 1986, pp. 22-42.

[BoBrLi76]

B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality," Proceedings of the Second International Conference on Software Engineering, 1976, pp. 592-605.

[BrAgBa76]

C. Brophy, W. Agresti, and V. R. Basili, "Lessons Learned in Use of Ada Oriented Design Methods," Proc. of the Joint Ada Conference, Arlington, Virginia, March 16-19, 1987.

[DoBa85]

C. W. Doerflinger, V. R. Basili, "Monitoring Software Development Through Dynamic Variables," IEEE Transactions on Software Engineering, vol. SE-11, no. 9, September 1985, pp. 978-985.

[Dy82]

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

[JeBa88]

D. R. Jefferey, V. R. Basili, "Validating the TAME Resource Data Model," Proceedings of the Tenth International Conference on Software Engineering, Singapore, April, 1988, pp. 187-201.

[Mc85]

F. E. McGarry, "Recent SEL Studies," Proceedings of the Tenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, December 1985.

[LoBa89]

C. Loggia-Ramsey, V. R. Basili, "An Evaluation of Expert Systems for Software Engineering Management," IEEE Transactions on Software Engineering, Vol. 15, no. 6, June 1989, pp. 747-759.

RaBa85]

J. Ramsey, V. R. Basili, "Analyzing the Test Process Using Structural Coverage," Proceedings of the Eighth International Conference on Software Engineering, London, UK, August 1985.

[Ro70]

W. W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," Proceedings of the WESCON, August 1970.

[SeBa88]

R. W. Selby, Jr., V. R. Basili, "Analyzing Error-Prone System Coupling and Cohesion," Technical Report TR-88-46, Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland, June 1988., IEEE Transactions on Software Engineering (to appear).

[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.

[WeBa85]

D. M. Weiss, V. R. Basili, "Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory," IEEE Transactions on Software Engineering, vol. SE-11, no. 2, February 1985, pp. 157-168.