Formal Methods in Systems Engineering

Edited by
Peter Ryan and
Chris Sennett

Springer-Verlag

d results from the use of Z in
g, Lecture Notes in Computer

nt Using VDM Prentice-Hall

nroom Software Engineering

Z to be published by Addison-

8 Software Quality : A Modelling and Measurement View

Victor R. Basili

8.1 Software Quality Needs

In order to deal with the concept of software quality, we need deal with such issues as quality definition, process selection, evaluation, and organization. In order to develop a "quality" object, we need to define the various qualities and quality goals operationally relative to the project and the organization. It is clear that there are a large number of qualities or characteristics that a product may possess and different projects will prioritize these qualities differently. Once we have defined the kind and level of qualities we want, we need to find criteria for selecting the appropriate methods and tools and tailoring them to the needs of the project and the organization in order to achieve these quality characteristics. This implies we may be interested in the quality of the processes as well as the quality of the product. We need to evaluate the quality of the process and product relative to specific project and organizational goals while the product is being developed, as well as when it is complete. This is to assure we know we are on the right track and can make the appropriate adjustments to the process to achieve our goals. Lastly, we must create a support organization to oversee the quality goals from planning through execution, by evaluation, feedback and improvement.

The software industry has been slow in recognizing these needs for software quality. Part of the problem has been a lack of understanding of the nature of the artifact and its desirable qualities and the difficulty of achieving the quality characteristics desired. In many organizations, the definition of quality is simply a small number of errors reported from the customer. This is a rather narrow and limited definition of quality since quality means more than error reports. Most companies have a software development methodology standards document (which few projects follow). For them, process selection means using this single model of process definition, independent of the project characteristics and quality needs. That is, using the same processes for a well understood project development and a first time product. Quality evaluation often means counting the number of customer trouble reports generated. This is a rather passive view of quality, not an active one. It does not provide a mechanism for learning how to produce software with the desired characteristics and does not provide an evaluation of the methods or tools needed to provide those characteristics. Lastly, for many companies the quality organization is the group that provides management with the trouble report data. This is not a quality oriented organization, quality is the result not the driver.

What kinds of quality approaches exist? We can have quality control and quality assurance. Quality control is the act of directing, influencing, verifying, and correcting for ensuring the conformance of a specific product to a design or specification. Quality assurance is the act of leading, teaching and auditing the process by which the product is produced to provide confidence in the conformance of a specific product to a design or specification. Quality control

is project oriented, quality assurance is process oriented.

Quality control involves such activities as evaluating products and providing feedback to the project. A quality control organization is typically part of the project and interactive with it. It requires people with a knowledge of the processes and the product requirements, and an understanding of the solutions.

On the other hand quality assurance usually involves the following activities:

- Definition and evaluation of processes
- Collection of data from quality control
- Feedback to the projects it is assuring, as well as to the organization and itself (for the purpose of learning).

It involves an independent chain of command from the project. People requirements involve a high level of knowledge with respect to technology and management and a deep understanding of the process and the product.

What control do quality control and quality assurance have? How often is a project stopped from moving on to the next phase? How often is a design rejected because it doesn't pass standards? The answer is not too often. This is because we do not have the information necessary to make the proper decision, i.e., we do not have models and baselines (measurement data) to provide the real understanding of the situation to exercise control.

8.2 Modelling Software Experiences

In the software business, we need to be able to characterize and understand the the elements of the business. We need to be able to differentiate project environments and understand the current software process and product. We need to provide baselines for future assessment. We need to build descriptive models of the business.

We need to be able to evaluate and assess our processes and products. That is we need to assess the achievement of quality goals and the impact of technology on products. We need to understand where technology needs to be improved, tailored. We need to be able to compare models.

We need to be able to predict and estimate the relationships between and among processes and products. We need to find patterns in the data, product and processes that allow us to build predictive models.

We need to motivate, manage, and control the software development process by providing quantitative motivation and guidelines that support what it is we are trying to accomplish. We need to build and use prescriptive models.

These issues argue for the need to define models to help us understand what we are doing, provide a basis for defining goals, and provide a basis for measurement. We need models of the people, e.g., customer, manager, developer, the processes, and the products. We need to study the interactions of these models.

These models should be as formal as we can make them, based upon the maturity of our understanding of the software process and products. Once we have models we can ask the following questions: Is the model correct in

principle? Does the model actually describe what we are doing? How can we improve the model based on theory, practice and analysis? How do we feed back what we have learned to improve the model or our adherence to it? We want to build descriptive models to explain what is happening. We want to define prescriptive models to motivate improvement.

What kinds of things can we package into models? At NASA/GSFC, in the Software Engineering Laboratory, we have built resource models and baselines, e.g., local cost models, resource allocation models, change and defect baselines and models, e.g., defect prediction models, types of defects expected for the application, project models and baselines, e.g., actual vs. expected product size, library access, over time, process definitions and models, e.g., process models for Cleanroom, Ada waterfall model, method and technique evaluations, e.g., best method for finding interface faults, products and product parts, e.g., Ada generics for simulation of satellite orbits, quality models, e.g., reliability models, defect slippage models, ease of change models, and lessons learned, e.g., risks associated with an Ada development.

Many of the models we use are based on measurement. Measurement takes on different forms. We perform objective and subjective measurement. Objective measures are absolute measures taken on the product or process, usually involving interval or ratio data. Examples include time for development, number of lines of code, work productivity, and number of errors or changes. Subjective measures represent an estimate of extent or degree in the application of some technique or a classification or qualification of problem or experience. There is no exact measurement and the measures are usually done on a relative scale (nominal or ordinal data). Examples include the quality of use of a methodology or the experience of the programmers in the application.

We use a mechanism called the Goal/Question/Metric, GQM [BaWe84], paradigm to build particular models based upon goals. A GQM is a mechanism for defining and interpreting operational, measurable software goals. It combines models of an object of study, e.g., a process, product, or any other experience model, and one or more focuses, aimed at viewing the object of study for characteristics that can be analyze from a point of view, e.g., the perspective of the person needing the information. This orients the purpose of the analysis, namely, to characterize, evaluate, predict, motivate, improve, specifying the type of analysis necessary to generate a GQM model relative to a particular environment. In this context, 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.

We must build models of our various products and processes. By way of example, consider the need to build an operational model of our education and training in a particular process. We articulate the process as the following set of activities:

1. Provide the individual with training manuals they must read.
2. Provide a course, educating the individual in the process.
3. Provide training by applying the process to a toy problem.
4. Assign the individual to a project using the process, mentored by an experienced method user.

5. After this the individual is considered fully trained in the process.

We convert into an operational model by associating a set of interval values with the various steps of the process. In this case, since the model is clear, each of the steps represents a further passage along the interval scale. Thus a value of "0" implies no training,

1. implies the individual has read the manuals
2. implies the individual has been through a training course
3. implies the individual has had experience in a laboratory environment
4. implies the process had been used on a project before, under tutelage
5. implies the process has been used on several projects.

Even though we call this a subjective rating, if the education and training process model is valid, then our model and the metrics associated with it are valid. Using the GQM, we generate a question that gathers the information for the model.:

Characterize the process experience of the team (subjective rating per person) : (again, "0" corresponds to no training)

1. have read the manuals
2. have had a training course
3. have had experience in a laboratory environment
4. have used on a project before
5. have used on several projects before

The data from the question can then be interpreted in a variety of ways, e.g., if there are ten team members, we might require that a minimum requirement is that all team members have at least a three and the team leader has a five, etc. This evaluation process will become more effective with experience over time.

Factors or characteristics that affect software development and which need to be modelled include people factors such as number of people, level of expertise, group organization, problem experience, process experience, problem factors such as, application domain, newness to state of the art, susceptibility to change, problem constraints, process factors such as, life cycle model, 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.

These models help us to classify the current project with respect to a variety of characteristics, to find the class of projects with similar characteristics and goals to the project being developed, and to distinguish the relevant project

environment for the current project. They provide a context for goal definition, reusable experience/objects, process selection, evaluation, comparison, and prediction.

The choosing and tailoring of an appropriate generic process model, integrated set of methods and techniques is done in the context of the environment, project characteristics, and goals established for the products and other processes [BaRo87]. Thus, the model for expressing process needs to provide a flexible process definition appropriate, information for process selection, and support for process integration and configuration, via tailorable definitions and characterizations for life cycle models, methods and techniques.

Goals need to be defined for the various processes. They help in the choice of a life cycle model, methods, and techniques. To show why this is necessary, consider the testing phase of a project development. What are the goals of the test activity? Is it to assess quality or to find failures? The selection of activities depends upon the answer. If it is to assess quality, then tests should be based upon user operational scenarios, a statistically based testing technique, and reliability modelling for assessment. If the goal is to find failures, then we might test a function at a time, general to specific. Reliability models would not be appropriate.

For example, it needs to permit advice and selection criteria based upon the problem characteristics and goals, containing such rules as:

- If the problem and solution are well understood, choose the waterfall process model
- If a high number of faults of omission expected, emphasize traceability reading approach, embedded in design inspections
- When embedding traceability reading in design inspections, make sure traceability matrix exists.

8.3 Model Evolution

The research in software engineering typically involves the building of technologies, methods, tools, etc. Unlike other disciplines, there has been very little research in the development of models of the various components of the discipline. The modelling research that does exist has centered on the software product, specifically mathematical models of the program function.

We have not emphasized models for other components, e.g., processes, resources, defects, etc. What is needed is a top down experimental, evolutionary framework in which research can be focused, logically and physically integrated to produce models of the discipline, that can be evaluated and tailored to the application environment.

In the SEL, we use an organizational framework called the Quality Improvement Paradigm [Ba85a]. It consists of the following steps:

- Characterize the current project and its environment with respect to models and metrics.
- Set the quantifiable goals for successful project performance and improvement.

- Choose the appropriate process model and supporting methods and tools for this project.
- Execute the processes, construct the products, collect and validate the prescribed data, and analyze it to provide real-time feedback for corrective action.
- Analyze the data to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements.
- Package the experience in the form of updated and refined models and other forms of structured knowledge gained from this and prior projects and save it in an experience base to be reused on future projects.

We have used this evolutionary paradigm to define, refine and evolve models in the SEL. For example, we have used it to help develop and formalize the definition of process models as follows:

- Use the project characteristics and goals to find the most appropriate process models for the current project
- Develop, modify or refine the process based upon the lessons learned from previous applications of the model
- Set goals to monitor those new or high risk areas
- Execute, collect and analyze data, making changes to the process in real time
- Based upon goals and analysis, write lessons learned and modify the process for future use

One such model has been the Cleanroom Process Model [Dy82], proposed by Harlan Mills. The key components of the process model are a mathematically based design methodology, the functional specification for programs, state machine specification for modules, reading by stepwise abstraction, correctness demonstrations when needed, and top-down development. Implementation is done without any on-line testing by developer. Testing is performed by an independent test group using statistical testing techniques, based on anticipated operational use. Testing takes on a quality assurance orientation, rather than a failure finding orientation.

Before applying the Cleanroom in the SEL environment, we ran a controlled experiment at the University of Maryland [SeBaBa87]. The goal of the study was to analyze the Cleanroom process to evaluate it with respect to the effects on the process, product and developers relative to differences from a non-Cleanroom process. The project was an electronic message system (1500 LOC) and there were 15 three-person teams (10 used Cleanroom). Each team was allowed 3 to 5 test submissions. We collected data on the developers background and attitudes, all on-line activity, and the test results.

The results of the Cleanroom project were quite strikingly in favor of Cleanroom. The Cleanroom developers felt they more effectively applied off-line review techniques, while others focused on functional testing. The Cleanroom developers spent less time on-line and used fewer computer resources. The Cleanroom developers tended to make all their scheduled deliveries. The product developed by the Cleanroom teams had less dense complexity, a higher percentage of assignment statements, more global data, and more comments. The Cleanroom products more completely met requirements and had a higher percentage of test cases succeed.

Based upon the success of the controlled experiment, we decided to experiment with the Cleanroom process model in the SEL. Following the Quality Improvement Paradigm, we

- Used the project characteristics and goals to find the most appropriate process models for the current project, i.e. we picked the process based upon the project needs and process strengths, e.g., Cleanroom for better lowering defect rate.

- Modified and refined the process based upon the lessons learned from previous applications of the model. Existing process model descriptions available for use included the standard SEL model, IBM/FSD Cleanroom Model, experimental UoM Cleanroom model. Lessons learned associated with the IBM/Cleanroom model included: basic process model, methods and techniques, and knowledge that the process was very effective in given environment. Lessons learned from the UoM/Cleanroom model included the facts that no testing enforces better reading, the process was quite effective for small projects, formal methods were hard to apply, require skill, and we might have insufficient data to measure reliability. We defined the SEL/Cleanroom process model using the best practices from our experience base models, e.g., informal state machine and functions, training consistent with UoM course on process model, methods, and techniques, emphasize reading by two reviewers, allow back-out options for unit testing certain modules, etc. When no new information was available, used the standard SEL activities.

- Set goals to monitor those new or high risk areas. Here the goals were to characterize and evaluate in general, and with respect to changing requirements.

- Execute, collect and analyze data, making changes to the process in real time. We monitored the project carefully and made changes to the process model in real time.

- Based upon goals and analysis, write lessons learned and modify the process for future use. We wrote lessons learned for incorporation into next version, redefining process for the next execution of the process model, and rewriting the process model definition.

Lessons learned from the first application of the Cleanroom process in the SEL included :

- the fact that we could scale up to 30KLOC and use the process with changing requirements.
- The failure rate during test reduced to close to 50% over our typical project.
- There was a reduction in rework effort: 95% as opposed to 58% took less than 1 hour to fix.
- Only 26% of faults were found by both readers.
- Productivity increased by about 30%.
- The effort distribution changed in that there was more time spent in design and 50% of code time spent reading.
- Code appears in library later than normal and more like a step function than in the standard project.
- There was less computer use by a factor of 5.

We also learned that we needed better training for methods and techniques and better mechanisms for transferring code to testers. Testers need to add requirements for output analysis of code. As expected, there was no payoff in reliability modelling due to the inability to seed a model with such a small number of failures. One of the side effects of the project was that the Cleanroom development caused more emphasis on requirements analysis and the requirements writers were willing to refine their methods.

Based upon the success of this project, two new Cleanroom experiments were defined. A goal for both was to try to apply the formal models more effectively, i.e., use Mills' box structure approach. One project involved a change in the application domain, keeping the size of the project about the same (30KSLOC). The second project was a scale up to over 100KLOC and the use of contractors.

8.4 An Organization for Packaging Experience Models for Reuse

Improving the quality of 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/modified to meet the needs of the current project (reuse) [BaRo91]. Systematic learning requires support for recording experience, and off-line generalizing, tailoring and formalizing of experience. Packaging 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 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, processes for learning, packaging, and storing experience, and the integration of these two functions. We define a capability based organization to deal with such needs, which differentiates between the software development (Project Organization) and the packaging of experience in models (Experience Factory) [Ba89]. The Project Organization develops products with the support of reusable experiences from the Experience Factory tailored to its particular needs. The Experience Factory is a logical 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.

8.5 Conclusions

The integration of the Improvement Paradigm, Goal/Question/Metric Paradigm, and Experience Factory Organization provides a framework for software engineering development, maintenance, and research. It takes advantage of the experimental nature of software engineering. It provide a framework for defining quality operationally relative to the project and the organization, a justification for selecting and tailoring the appropriate methods and tools for the project and the organization, a mechanism for evaluating the quality of the process and the product relative to the specific project goals, and a mechanism for improving the organization's ability to develop quality systems productively.

References

[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.

[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

[BaPa85] V. R. Basili, N. M. Panlilio-Yap. *Finding Relationships Between Effort and Other Variables in the SEL* IEEE COMPSAC, October 1985.

[BaRo87] V. R. Basili, H. D. Rombach. *Tailoring the Software Process to Project Goals and Environments* Proc. of the Ninth International Confer-

ence on Software Engineering, Monterey, CA, March 30 - April 2, 1987, pp. 345-357.

[BaRo91] V. R. Basili, H. D. Rombach. *Support for Comprehensive Reuse* Software Engineering, IEE British Computer Society, September 1991.

[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.

[Dy82] M. Dyer. *Cleanroom Software Development Method* IBM Federal Systems Division, Bethesda, Maryland, October 14, 1982.

[Mc85] F. E. McGarry. *Recent SEL Studies* Proceedings of the Tenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, December 1985.

[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.

9 Modellin

Dan Craigen

9.1 Descriptio

One of the central
of digital systems.
digital systems can
Hence, there is intere
model the digital en
of models should lea

9.2 Discussion

The following list o
participants prior to
clear that the topics

Topic 1: How does
of digital systems?

- What are the
them?
- What are the
- What are the
them?
- What is requi
digital system
- Identify genera

Topic 2: What insig

- Are we develo
primary defini
- Identify genera

Topic 3: What can w
we have now?

- Are the curre
If not, how is
those models
and explicit?

As digital systems grow in size and criticality, means must be sought to master the resulting complexity. One such technology goes by the name "formal methods", where this is best read as the application of mathematical techniques to the problems of developing quality systems. Though powerful, these techniques will doubtless find their best impact when used in conjunction with other systems engineering techniques.

In this volume a number of practitioners in the field address the issue of how best to integrate formal methods with other systems engineering methods, in particular quality assurance and structured design methods. Much of the material draws on practical experience gained in applying formal methods to "real", industrial-scale projects.

Also included are two chapters discussing some of the social implications of the use and role of formal methods.

Two appendices provide surveys of formal methods tools and applications, from North America and Europe.

This book provides essential guidance to anyone interested in the potential (and limitations) of this important technology in addressing the so-called "software crisis".

ISBN 3-540-19751-6
ISBN 0-387-19751-6