



Driving Quality Through Parametrics

Daniel D. Galorath, Lee Fischman, and Karen McRitchie
Galorath Incorporated, The SEER Product Developers

In this article, we show how prediction models are used to improve delivered quality. We further show that if you can anticipate and plan for the factors that affect quality, you can leverage quality management activities to improve the entire development effort.

Software development models¹ are gaining acceptance in the software project estimating community, which is always challenged to establish cost and schedule objectives before projects begin. Predictive models can in fact be further deployed into software projects to improve the quality of development.

Developers implicitly understand the notion of software quality; however, many ideas about quality unfortunately go no farther than active prevention such as testing or walk-throughs.

As critical as active quality control is, good planning will multiply the effectiveness of any effort, saving both time and money. But how can plans be laid without fully anticipating the factors affecting quality? In the haze of battle surrounding most development efforts, software development models provide the answer.

What Is Quality?

Discussions about software quality all too often focus on a single measure: defects delivered. Indeed, this may be the most significant measure of quality because software is useless—or worse—if it suffers from too many bugs. However, as with any other product, there are many dimensions to software quality.

- **Correctness.** Is the program correctly specified?
- **Usability.** Can users learn to use the software with reasonable effort?
- **Efficiency.** Does the software minimize the use of hardware resources?
- **Reliability.** Is the mean time between failures sufficiently long?
- **Adaptability.** Can the software be easily adapted to new uses?

Table 1. Cost/schedule/defect trade-off report.

	More Testing	Less Testing	Difference
Development Schedule Months	31.54	29.89	6%
Development Effort Months	1,422.39	1,210.15	18%
Development Base Year Cost	\$20,909,062.00	\$17,789,243.00	18%
Defect Prediction	33	60	-45%
Constraints	MIN TIME	MIN TIME	
Peak Staff	63.59	57.10	11%

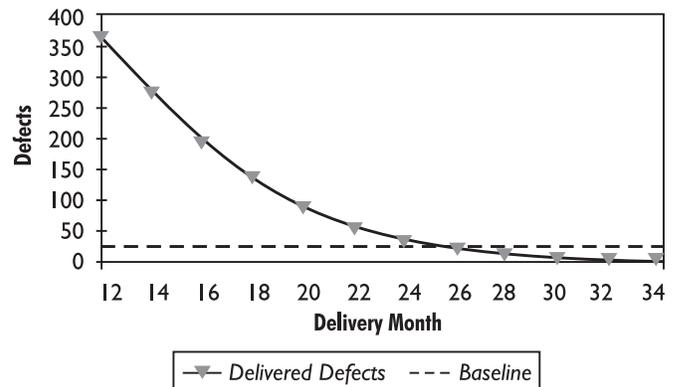


Figure 1. Defects delivered vs. length of development.

- **Robustness.** Can the software be stressed without breaking? Does it stand up to intentional or negligent user abuse?
- **Maintainability.** Once delivered, how challenging is it to maintain the software?

Software development models can directly account for many of these quality factors, either directly through estimates, i.e., defects delivered, or via parameter settings that in turn drive estimates.

An Overview of Parametric Models

Parametric models allow developers to specify software project variables and to receive in return estimates of effort (cost), schedule, and defects. Variables typically include complexity of the software to be developed, specification and test level, quality of the development staff and tools, complexity of the development language, and software size. Vendors of more mature tools have had a longer opportunity to collect data and perform enhancements, so more variables are generally available for their models.

Parametric models have several advantages over other methods of prediction. First, vendors work continuously to assure that their tools are accurate. The better tools also can be substantially calibrated to the specifics of an organization while retaining the essential sensitivities of key parameters. These tools give rapid, elaborate feedback and therefore can be used for realistic trade studies, even in a collaborative mode with “heads up” conferencing features.

For the concurrent engineering necessary to simultaneously satisfy cost, schedule, requirements, and quality goals, the benefit of parametrics is clear. No other method permits such rapid, elaborate interaction between varied

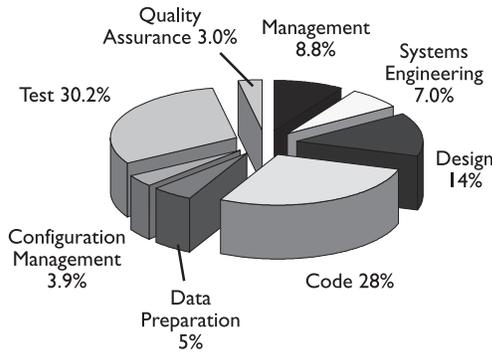


Figure 2. Activity allocation chart.

interests. Once development goals are set, parametric estimates can be used by developers to ensure that quality goals are achieved at least cost.

Defect Prediction

Of the many different aspects of quality, delivered defects are among the most obvious and quantifiable. A number of defect prediction methods are in use that rely on gross volume and complexity metrics such as size,² Halstead Software Science Volume, McCabe Cyclomatic Complexity, or other composite measures.³ An “integrated” defect model allows defect predictions to be evaluated alongside of staffing and cost considerations, which opens up a world of comparative scenarios.

The most useful prediction for a quality model is *delivered defects*, meaning those that escape detection and are delivered to the end user. A defect prediction allows you to plan for acceptable magnitudes and take corrective actions—follow a better process, lay on further testing, reduce scope, lengthen schedule—when predictions are too high.

Table 1 illustrates a trade study driven by defect predictions. For the testing effort required to halve delivered defects, costs will rise somewhat, and schedule less so.

An alternative to predicting delivered defects is modeling *potential defects*. Doing this allows developers to engage in explicit defect-related “what-if” scenarios, such as illustrated in

Figure 3. Development and maintenance effort as quality assurance increases.

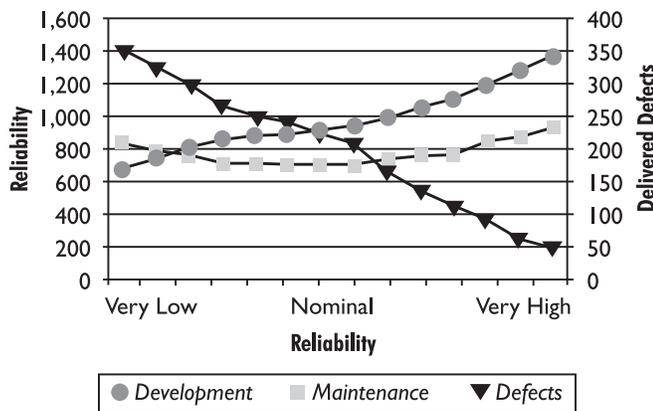


Figure 1. The dashed line shows the defect level given an optimal schedule that minimizes development effort. If the product is delivered earlier, defects will rise above this level, but if they are delivered later, defects will be lower. Note how the defect penalty decreases as the development schedule stretches. This is a powerful tool for managers to have to plan schedules to specific defect targets.

Modeling Promotes Active Quality Control

It makes sense not only to predict defect levels but also to predict adequate levels of quality control. Parametric models offer developers the advantage of a database of completed projects and industry wisdom. They show in concrete terms exactly how many employees are required to deliver a product of certain quality.

Models provide insight into quality control activities by parsing effort and staffing estimates into individual labor categories and activities. These parsing factors can often be ad-

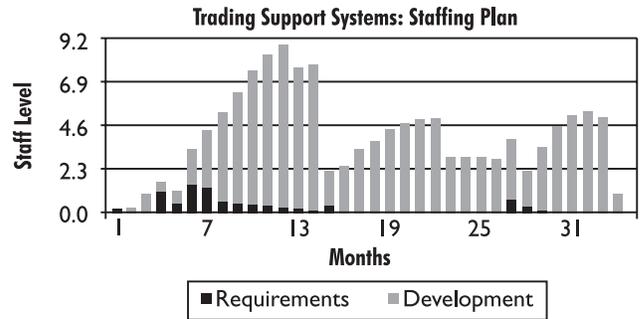


Figure 4. Staffing profile for a large project.

justed to the organization and the development process. This not only makes staffing estimates more suitable to specific development environments but also allows organizations to better promulgate desired levels of quality control.

Is it necessary to rigorously follow the testing level indicated? The answer is that they are benchmarks only, indications of what past development teams have required in order to achieve defect efficiencies along the lines of those envisioned. Maybe, quality targets can be achieved with different test staffing from that indicated; maybe, they cannot.

Accounting for Other Quality Factors Through Specification

The relationship in modeling is “many to few”—many parameters are available in a parametric model to specify factors from the development environment to the final product. The predictions that result are usually limited to the trinity of cost, staffing, and defects.

This mix of estimates and parameters allows developers to account for many quality factors, as indicated by estimates or as specified by parameters. With parameters, the analyst is not *predicting* quality, as in the case of defects, but rather is *specifying* quality, then judging the impact on cost, schedule, and defects. Aspects of quality handled via parameters include efficiency, adaptability, robustness, and maintainability.

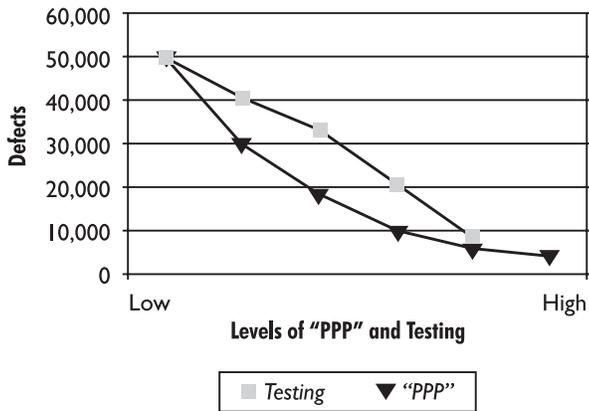


Figure 5. Impact on defects: team quality vs. testing.

As an example, maintainability is strongly correlated with maintenance costs, which can be modeled by varying other parameters. Figure 3 shows how reliability levels, for example, impact not only upfront development costs but also eventual maintenance costs. As specified reliability increases, defects decline, but development costs increase. Maintenance costs may also decline with increasing reliability, but soon the specification for a system becomes so rigorous that maintenance costs also rise.

Some quality-related attributes are not specified with parameters but can be accounted for in other ways. For example, usability may impact size, which is a prime input to a development model. Cost and schedule estimates will vary in direct proportion to software size, matching the intuitive result that greater quality comes only at a higher price. Development models tell you exactly how high that price will be.

Improving Quality by Modeling the Development Process

The sheer utility of development models' planning features offers other avenues toward improving quality. With insight into your project, as Figure 4 suggests, you are much closer to engineering quality into your development process.

All too often, quality slips when staffing requirements are poorly anticipated. Knowing the optimal staffing profile for a project improves planning, lessens staffing-related volatility, and therefore permits the timely application of testing activities. This particular chart also makes explicit the proper mix between early requirements work vs. coding and testing; as is well known, sufficient requirements definition does more to determine quality than testing.

Development models simulate reality by incorporating known development dynamics. For instance, it is less expensive to do good work first than to apply more stringent testing later. Figure 5 illustrates such a trade-off between the three P's of "people, process, product" vs. the alternative of increased testing. The impact on defects delivered is shown; either curve is drawn holding the other factor constant. Notice how there is a disproportionate return on improvements in the develop-

ment team, whereas there is only a linear return on improvements in testing.

Conclusion

Although parametric models have long been used to establish cost targets, they can be used for much more. Modern software development models have years of analysis support invested in them so that they can address such dynamic management issues as quality. If your organization uses parametric modeling for its estimates, see your estimators to see what they and their tools can do for you. ♦

About the Authors

Daniel D. Galorath is president of Galorath Incorporated. He has worked in all aspects of software development and software management and is one of the principal developers of the SEER-SEM software evaluation model. His published works include software cost modeling, testing, error prediction and reduction, and systems requirements definition.



Lee Fischman is special projects manager at Galorath Incorporated. He is a frequent consultant on estimating projects, and he also conducts core research and development of SEER software tools. He wrote the Software Evaluation Guide for the Office of the Secretary of Defence, Program Analysis and Evaluation, and he has explored software economics and estimating in numerous papers over the past several years, all available at <http://www.galorath.com>.



Karen McRitchie is vice president of development at Galorath Incorporated, responsible for design and development of SEER tools. She has nearly 10 years experience in software and hardware cost-estimating and hardware reliability modeling. She has been a lead member of several Air Force cost-estimating teams for major Department of Defense programs and has taught dozens of estimation methodology courses for costing professionals.

Galorath Incorporated, The SEER Product Developers
Voice: 310-414-3222
E-mail: support@galorath.com

Notes

1. Mathematical estimation models are known to the cost estimating community as "parametric models." As understood in mathematical English, this implies that functional forms are pre-specified. However, to costing personnel, parametric means only that these models have parameters to modify; no comment is being made about functional form.
2. Lines of code, function points, and object-based metrics are the most commonly used size measures.
3. For a description of how SEER-SEM handles defect prediction, refer to the "SEER-SEM Defect Prediction" technical note available at <http://www.galorath.com>.