

# Using Tolerance Bands on Test Related Metrics to Plan and Manage the “Crunch”

**Earl Burba and Jon D. Hagar**

Lockheed Martin Astronautics

Denver, Colorado, 80201

jon.d.hagar@ast.lmco.com

(303) 977-1625

## **Abstract:**

This paper examines the successful application of tolerances on process metrics used in management activities of an ongoing test project. Statistical process control (SPC) is not new to environments such as a factory production line, but use of SPC and tolerances in software engineering is not well established. We report the use of tolerance bands about a process plan and associated measurement. The paper presents the implementation with some samples from our actual project. The application of tolerances has worked for us in controlling processes. We experienced a “smoother road” through the use of tolerances during schedule rush during the end of efforts. Our team has been comfortable with the technique since they allow metrics with flexibility. The processes were managed and the system could handle some variation. The result was a more stable system during the “crunch” at the end of the test cycle.

## **1. Introduction**

For several years our Independent Verification and Validation (IV&V) project at Lockheed Martin Astronautics in Denver, Colorado has used metrics in a variety of ways to support the management, testing, and analysis of real-time embedded flight software. The software under test is responsible for the guidance navigation and control of rocket systems. With the changes in software standards, customer focus, increasing software complexity (and associated issues), and failures like the Ariane 5 rocket system {1}, the interest in and criticality of our IV&V activities has increased. We have always had varying kinds of metrics, but the scrutiny directed at them has now increased.

In planning and metrics work in IV&V, we asked ourselves a variety of questions. While management and customers are interested in metrics and many tools produce them, what do the working engineers, first level supervisory staff, and management do with them? What things can be done to benefit from them and improve their frequency of use? What metrics contribute to successfully planning and controlling a software testing project?

We have observed that there are a large number of metrics. Today, every analysis tool seems to produce some set of metrics. And yet their true utility is often questioned. For us in our practical world, a metric must be considered from the standpoint of usefulness in controlling a product or process. This paper relates the recent experience and use of software metrics for technical and management purposes on our project. We have observed the following:

- Metrics can be overwhelming in terms of numbers (one tool we use can produce hundreds of metrics) and must be considered in a hierarchical fashion.

- Metrics that are used should change with program phase and activity.
- Metrics should be viewed differently depending on the perspective and should not be treated by a user as an absolute. Three possible views are customer, management, and engineering, though customers and managers can view metrics from very similar perspectives.

Our IV&V objectives were to produce quality software on schedule and within cost. We also wanted to manage the schedule pressures that inevitably happen during a project (particularly at the end when the project is coming down to “wire”). We selected process related metrics to support those objectives. We have found that a few select metrics can solve the metric overload problem and allow in-use measurements to change over time to best accommodate objectives. The observation of “keep things simple” is not new. However, overcoming the tendency to treat metrics as absolutes required some innovation for us. In our case, we introduced the use of tolerance about the measure to make a metric more flexible to engineers while at the same time increasing usefulness to overseers.

In this paper/presentation we outline our situation, and we identify our use of tolerances about a metric citing data from several program areas. The paper/presentation then examines the pros and cons of this.

## **2.0 Views of Metrics: Management and Technical**

Our product area at Lockheed Martin Astronautics in Denver, Colorado, tests critical software systems used in flight control systems of booster rockets. Production systems are usually one of a kind that must work the first time or hundreds of millions of dollars may be lost. These systems are typically very complex, consequently failures or errors could be introduced from many sources. These software-systems have the following characteristics: real-time; spacecraft/booster flight control; minimal human intervention possible; and numerically intensive calculations of such critical items as, trajectories, flight dynamics, vehicle body characteristics, and orbital targets. Development programs are small—under 50,000 source lines of code. We test from a unit level to an integrated level which includes much of the actual hardware driven by large simulations. Overall, this approach and tool set has been successful {2} in taking input development products, doing IV&V, and generating test results (reports). In spite of success, we look for improvements in our processes and tools to save time and money. We have found that metrics aid in gauging improvements, measuring performance, understanding test progress, test planning, calculating cost performance, and process control including risk management.

Our team views metrics from at least two view points: management and technical. These views are associated with the different stakeholders: supervisors, project management, planning personnel, accounting, and customers. Much of the team tends to focus on the process related {3} metrics, concentrating on performance, cost, and schedule. In management, we measure things such as number of tests generated, number of requirements actually closed versus what was planned, cost in hours of doing work versus planned budget.

Engineers use the process metrics in statusing efforts, but are often more interested in the technical considerations which can be correlated to product metrics {3}. Engineers look at the size of the module, number of comments, complexity of the code, number of functions, and number of errors that have been found. We have found that hard and fast rules such as “a McCabe {4} complexity factor of less than ten shall be maintained”, need to be more of a guideline than a rule. We have several commercial tools that produce product metrics on our code, which in one case is Ada. The tool, Adatest {5} produces over 30 metrics every time we run a test, while the tool AdaMat {6} can produce hundreds of measures. These are used in test inspections as well as a starting point for designing tests and test completion (number of paths covered). Additionally, these metrics allow us to do better estimating and planning which is then fed into our process performance metrics. So we have some overlap between the views of metrics.

Each view of metrics is legitimate. As part of program efforts, we began tracking metrics and then using them to plan and monitor progress. We did this by establishing a metric and projecting it into the future (a plan). Then we charted actual progress and compared it with our projected estimate (see Figure 1). This led to instances where the actual numbers did not match (fell below) the planned numbers. The team then became “concerned” about performance. This led to analysis, planning and recovery efforts. Many times a deviation from the projected metric was just a normal variation in production. At the end of planned efforts, when we completed schedule plans, we found that there was minimal reason for concern” about these “divots”. Once we realized this, we came up with an idea: Use tolerance bands around the projected metric to account for these variations in the system. We found it was better to use metrics as a quick reference point or as an indicator of some quality.

### **3.0 Using Tolerance on a Process Metric**

Tolerance is not new in the production environment, but the application of tolerance to process metrics in a software engineering/test environment is just being established as reported in this paper. Statistical process control (SPC) has been advocated for use in production environments for years. Often in SPC, upper and lower control limits are employed around a norm {7}. However, SPC is usually applied in factory and production environments where some known physical characteristic or large production rate is being measured. The application to engineering (intellectual) activities and small lot products is less well understood or practices {8}, though written about in areas like the “personal software process” {9} and Statistical Methods for Software Quality {10}. Activities like the Software Engineering Institute Capability Maturity Model (SEI CMM) and ISO 9001, also contain elements that identify the use of statistics. In the CMM, standard core metrics {11} are identified but there is minimal guidance or accepted standards on actually applying these. Users are left with direction to have metrics, without the supporting techniques such as SPC provides in the factory production environment.

In IV&V management, we decided to use metrics as a planning, scheduling and tracking tool. To do this, we established an estimated planned measure on a key process we were

doing. The plan was based on historic production rates as well as estimates. We then defined high and low bounds around the planning measurement (see Figures 1, 2, and 3). Then we tracked and measured our actual production rates for each time interval. As long as our actual production rate was within the tolerance bounds, we determined that our process was within what we termed “normal” variance of our plan or in control. No significant action by management or concern by customers was warranted until we hit a tolerance limit. Once a tolerance was reached, some effort like re-planning or modification of a process became necessary to achieve completion.

### **3.1 How Do We Calculate a Tolerance?**

This question does not have a well refined answer. We ended up doing many tolerance estimates using engineering judgement to do the upper and lower curves. A few tolerances were created by using a time varying percentage on the projected “nominal” metric (example 5%, 10%, 15%, and 20% over time of the nominal line over time) or a timing varying percentage based on allocated manpower. We would like to have better techniques for calculating the tolerance. Items to consider would be time varying functions, possibly parameter based on actual historic data, which we did not have since the in-use metrics had no historic base. Additionally, we considered of methods to normalize the data based on product metrics such as complexity. But, we have not yet implemented these ideas and research is still needed.

### **4.0 Application Of Tolerances Around A Process Metric**

Presented in this section are some samples from our use of metrics with a tolerance. The actual data presented includes the number of requirements tested, number of software programs generated, and number of units tested (unit testing). The data was from an ongoing test effort and represents both work that has been completed and things, that at the time of the writing of this paper, were actively in use. Additionally, we have other metrics in use that we have considered for tolerance metrics but have not established the upper and lower limits yet. These include software problem report rates and product complexity levels (like Halstead and McCabe).

### **4.1 Requirements And Functions Tested**

One measure of a test group’s progress is to count the number of individual requirements under test, and then track the testing to these. As a test demonstrates one or more requirements, a requirement can be said to be tested or closed. Once all requirements have been tested, this functional type of testing can be considered completed. There is always a debate about what an individual requirement is, but for the purpose of this discussion, the definition is not the issue.

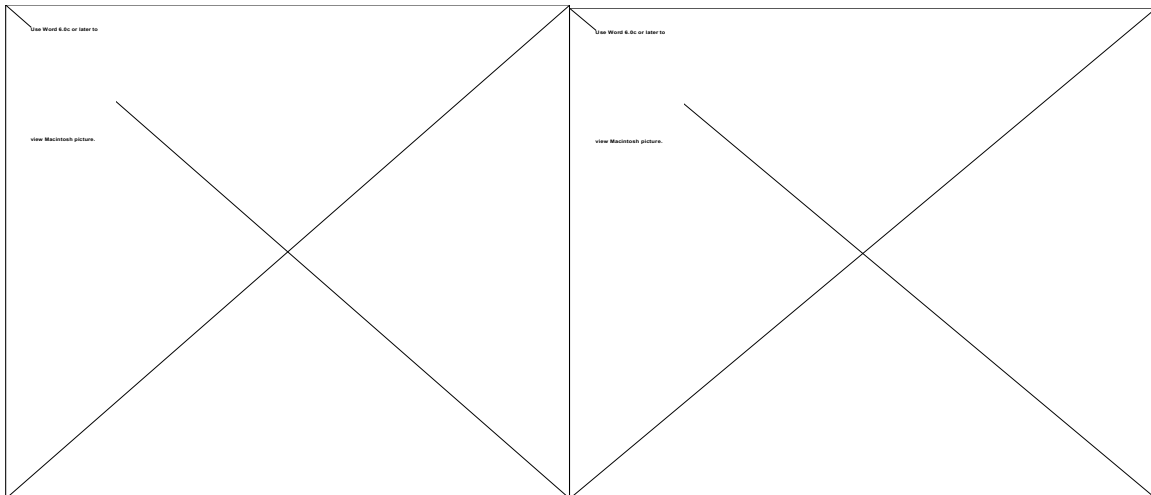


Figure 1 - Requirements Closed by Testing

Figure 2 - Test Tool Production

In Figure 1, each time period (month) has a planned number of requirements to be tested. This is represented by “Nominal Plan”. Above this is a line indicating an “Upper Limit” scenario. When we achieve more requirements closed than initially projected, we must ask ourselves such questions as, “Are we being as complete as we’d planned or did we just over scope the job (allowing for better planning next time)?” The “lower limit” is a minimum production rate that must be maintained and indicates the point above which our test team must stay to avoid corrective action. When we reach or exceed this minimum line, re-planning, additional budget, reduced test scope, or other action may need to be considered. If we are within the upper and lower lines, we consider that the testing is progressing within acceptable limits. The limits are large but were established based on estimated minimum and maximum production rates. In Figure 1, we can see actual progress is within tolerance. We can also watch the trend line (extension of the actual slope) and determine at what point in the future we may cross a tolerance. This allows a guess at the prediction of when an effort might get into trouble.

Finally, to account for requirements change traffic, we allow at the end of the project, a range of requirements (between about 1600-1900) on the upper and lower limits. This is our plan and we expect our actual number of requirements closed to be within this range. Then, we track the “Current Total RQMT Count”, which is the number of actual requirements at the current time point. We will be 100% complete when the actual production count equals the “Current Total RQMT Count”. And, as long as the “Current Total RQMT Count” falls within range of 1600-1900 at the end, we do not need any additional planning due to requirements growth/reduction. This approach allows us to expand or reduce numbers of requirements to be tested without having to “rebaseline” plans.

#### 4.2 Number Of Software Programs Generated

In support of testing, we generate test support software. These software programs generate things like inputs to tests, analysis of test outputs, static analysis, or direct support of test execution. The generation of the test support software adds significant efforts to any test program and can be a major source of cost or schedule time.

During the development of these test programs we established a measure on their production (similar to units of code produced or functions generated). We did not measure lines of code, instead we measured whole programs that satisfied one or more functions. So, in Figure 2, each item being measured is a standalone program of varying size, function, and complexity.

We again established a planned production rate based on software characteristics. Next, we estimated the tolerance of what our high and low rates of production might be (upper and lower lines). We then tracked our actual production rate. In this case, we reached a “lower” tolerance in *February*, and had to analyze why. In this example, it turned out we had underestimated the size and complexity of a couple of programs, thus they took several months longer. This impacted when we could start other programs (resources weren’t available), which “blew” the metric. We solved the problems by working overtime, getting some added resources, and rescheduling some activities to eliminate resource and/or time constraints on the critical path (note: the graph of Figure 2, where there is a reduction in total tool count). This allowed the completion of test support programs “on time” relative to when we needed them. This also pointed out the observation that metrics often must change and evolve with a program to be really useful. Have in use metrics allowed better control to avoid and/or minimize the final crunch at the end. Casting them in stone may mean they do not get used.

#### **4.3 Number of Software Tests (Units) Generated**

This metric tracks the number of test procedures generated. A test procedure may test one or more requirements, functions, or units of code. In this case, each test covers one unit of code. Each unit is tested at test levels of: 100% statement executed, 100% branches taken, and 100% decision coverage. Additionally, each unit is tied to one or more design level specifications.

In Figure 3, the “plan” line indicates the number of tests allocated to each increment (month) of time. It is not unusual for tests to take differing amounts of time based on the complexity of the unit and the number of requirements implemented. This means that not all tests are equal in terms of such things as effort. Engineers used product metrics like Halstead and McCabe {4} to estimate when a module could be completed. These were totaled over time to produce the plan line. We then applied a tolerance to get high and low production rates in the same fashion as the requirements tolerances used. In this “in work effort”, we can see that the “actual” line is trending below the planned line. Some action will be needed by the team when the actuals cross the “must meet” line.

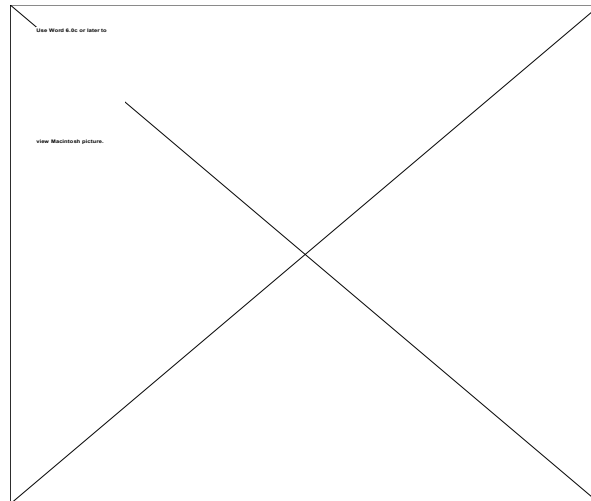


Figure 3 - Module Test Cases

### 5.0 Our Experiences

We have applied these concepts in several situations with positive results from both views, management and technical, of a metric. People with either view found:

- Up front planning is important.
- Allocation of metrics so that all planned tests are covered by a metric is important.
- Creating metrics with an “owner” (someone who is responsible for it ) is important.
- Coordination between owners is important, so the right testing is done.
- Have automated methods for collecting metrics.
- The approach allowed inevitable deviation from the nominal baseline.
- The use of tolerances established “hard” threshold lines that once exceeded indicated the need for some management and/or engineering action.
- Tolerances accounted for the inevitable variation in a process and minor schedule slips.
- This system provided a metric based planning tool that engineers were comfortable with and met management/customer needs for visibility into the engineering activities.
- Re-test or requirement growth factors (as shown in Figure 1) could be built into the metric/metric plan.

Overall metrics with tolerances contributed to a better working environment. We also found that the team did less re-planning, accounting, justification, and other activities related to dealing with a metric that wasn't exactly met at a given time point. We still had “management by the numbers”, but the system became more stable. We were able to use metrics to assist in planning and status tracking by providing information in a format that was useful to all team stakeholders. And as the life cycle progressed use of trending and exceeded tolerances allow management of the “crunch” before the end because process, resources or other changes could be introduced early enough to make an impact.

There were areas we found a need for improvement in the future. These were:

- Process metrics can indicate trouble early to avoid later failures and the “crunch”.
- The planning function is an estimated line that needs to be more closely coupled to historic data. No attempts were made to use any kind of standard parametric model since we felt that this did not apply to our process details.
- The tolerances were done mostly by “hand”. No formal method was known or found to allow a “scientific” balancing of upper and lower control limits, other than planning estimates.
- The inclusion of requirement growth factors (or additional “lines”) caused confusion which needed added explanation. We found the rule of “keep it simple” applied so that the users of a graph were not overwhelmed.
- In general, the application of things like SPC to engineering/software process metrics is not well understood or documented. Additional research on the application of tolerance to metrics as viewed by technical and management people in software should be considered by the research community.
- Metrics can mislead if you are measuring only parts of the process or if they are measuring the wrong thing.
- Metrics and measurement plans potentially will change over time and this should be part of the planning and control process.
- Choose a few simple metrics and the “buy in” by the stakeholders.

## **6.0 Conclusions and Summary**

We have outlined a successful application using tolerances about process metrics which are used in management activities on an ongoing project. We presented some examples from our actual project. Summary observations include some of the limitations of what we did, lessons learned, and successes. The application of tolerances has worked for us in controlling processes as related to scheduling. We learned it was not enough to have a metric but it must be integrated with up front planning, situation analysis, rescheduling, engineering and management. We have experienced a smoother road in the beginning, middle, and end by using tolerances. Our engineers, customers and managers are comfortable with the technique. The process was still managed, but the system could handle variation without immediate over reaction and consequence. The result was a more stable system of planning and control.

There is a need for more research into how to calculate and apply SPC type metrics to engineering and software processes. Additional validation also seems in order. We may not need more measures, but better techniques that employ what we have. This might allow more uses of metrics in a productive fashion.

## **References**

1. J. L. Lions Et Al, ARIANE 5 Flight 501 Failure, Report By The Inquiry Board, Paris, 19 July 1996.
2. Jon Hagar, Lessons Learned From Incorporation Of Commercial Computer Aided Software Engineering Tools In A Flight Critical Software Test Environment, Proceedings From AIAA/IEEE Digital Avionics Systems Conference, Oct. 1996, Pp. 125-131.
3. IEEE Std 982.1-1988, IEEE Standard Dictionary Of Measures To Produce Reliable Software, June 9, 1988.
4. Introduction To McCabe Tools Manual, McCabe & Associates, 1995
5. Adatest Reference Manual, Information Processing Limited, 1995.



6. Adamat Users Manual, Dynamics Research Corporation, 1991.
7. Statistical Process Control Manual, American Society For Quality Control, Automation Division, 1986.
8. Variable Reduction Process, The USAF R&M 2000, 1994.
9. Introduction to the Personal Software Process, Watts, S. Humphrey, SEI Series in software engineering, 1997.
10. Burr and Owen, Statistical Methods for Software Quality, International Thompson Publishing inc, 1996.
11. Paulk et al, Capability Maturity Model For Software, Software Engineering Institute, Carnegie Mellon, 1991.

## **BIO**

**Earl Burba**

**Jon Hagar**

Lockheed Martin Astronautics Company

Mail Stop H0512

P.O. Box 179

Denver, CO 80201

303-977-1625

303-977-1472 (fax)

hagar@den.mmc.com

### **Authors:**

#### **Earl Burba Biography**

Mr. Burba is a staff software engineer with Lockheed Martin Astronautics in Denver, Colorado. He is currently assigned to verification testing and software test tool programming. Mr. Burba has a B.S. in computer science from Colorado State University, Ft. Collins, Colorado, and a Masters degree in C.I.S. degree from the University of Denver, in Denver. Mr. Burba has over 12 years experience in software test of real-time interrupt and event driven flight software. His experience lies mostly in space related projects, supporting the Magellan mission, Inertial Upper Stage, Transfer Orbit Stage, Global Geospace Science, etc. Mr. Burba's interests include WWW technologies verification, validation, IVV, software testing, and software engineering.

#### **Jon Hagar Biography**

Jon Hagar is a lead software engineer supporting software verification and validation testing at Lockheed Martin Astronautics in Denver, Co. He has a B.S. in math with specialization in civil-engineering and software from Metropolitan State College of Denver, Colorado, and an M.S. in computer science with specialization in software engineering and testing from Colorado State University in Ft. Collins, Colorado. Mr. Hagar has worked in software engineering, particularly testing/verification and validation, for more than 15 years. He has supported primarily booster and space related projects. Mr. Hagar has experience in the software domain of real-time, reactive embedded control systems as well as test software development using numerous languages, including JOVIAL and Ada. Mr. Hagar is a member of ACM and IEEE and teaches classes at Lockheed Martin Astronautics., He has published articles on software reliability, testing, formal methods, and critical-systems, as well as presented papers at NASA and Software Productivity Consortium (SPC) working groups. Mr. Hagar's work interests include software testing, verification, validation, system engineering, reliability, neural-networks/GAs in testing, test support tools, and quality assurance.