

Actionable Analytics for Strategic Maintenance of Critical Software

An Industry Experience Report

Dan Port, University of Hawaii at Manoa

Bill Taber, Jet Propulsion Laboratory

// JPL software engineers have used a robust software metrics and analytics program that enables actionable strategic maintenance management of a critical system in a timely, economical, and risk-controlled fashion. //



AT NASA'S JET Propulsion Laboratory (JPL), the Mission Design and Navigation Software Group (MDN) has two critical systems in

operation—a legacy navigation system and its replacement, Monte. These systems are in continual operation for most of NASA's

deep-space missions. The development and maintenance of them is unquestionably critical to the success of those missions. For many of these missions, encountering a defect at a critical time, waiting for a new feature or enhancement, or known defects not repaired in a timely manner not only inhibit operations but also may risk the loss of a billion-dollar mission. It is simply too risky to have a critical system crippled or nonoperational for an arbitrary period. From years of experience, we have learned that poorly managed maintenance can lead to mission failure.

With over 800,000 lines of continually evolving code, we don't know exactly what or when maintenance issues will surface, only that inevitably, they will, and that we must plan to resolve them in a timely manner. The reality is that we are continually faced with exceptionally tight resources and schedule constraints. It is impractical to have a standing army of qualified maintenance staff at the ready to attack bugs and implement system enhancements and adjustments on demand. With these constraints and the inherent uncertainty in the demand for maintenance, it is unimaginable how we could have successfully sustained Monte over the past 12 years without reliance on our data models and analytics.

We use models and analytics to monitor our process, report confidence and risk in system quality, forecast effort, and make a strong business case for budget and staffing levels. These actions depend on reliably knowing how many defects we may encounter, the likelihood of their surfacing at a critical moment, and how long it will take to discover and correct them. We also must

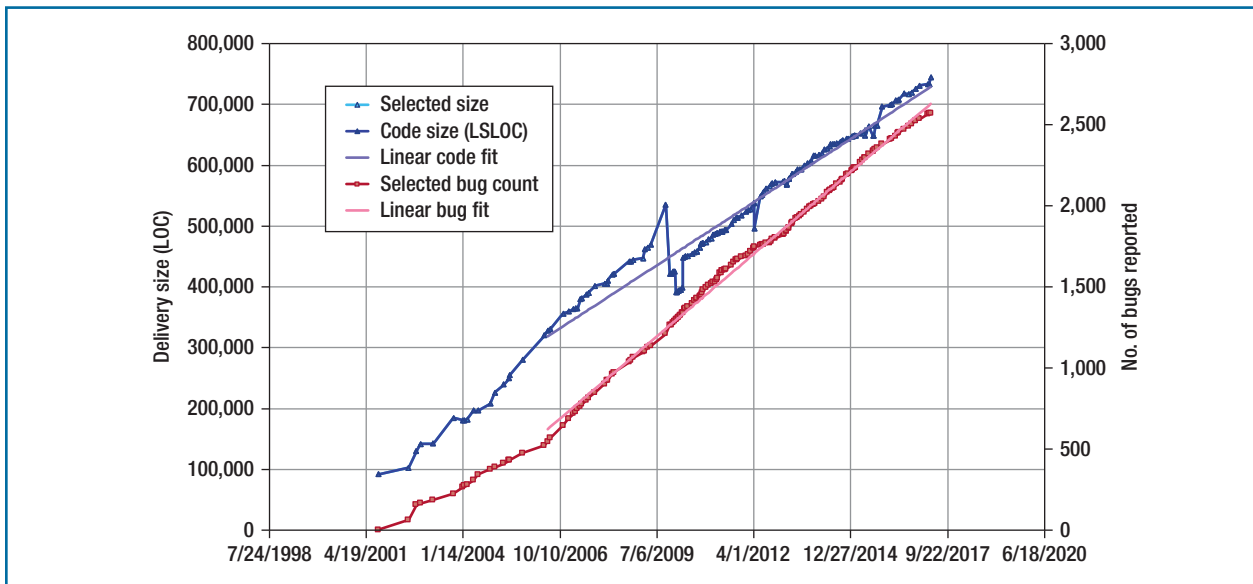


FIGURE 1. Code size and defects reported over time for the Monte navigation system, by release.

account for enhancement requests, when they will be needed, the additional maintenance burden (i.e., new defects) they will impose, and determining the appropriate staffing size needed to meet the institution's risk tolerance for continuously operational ground navigation software.

These issues are challenging and require having credible and defensible maintenance models and authoritative analytics to address them. This report presents some examples of MDN's use of a robust software metrics and analytics program that enables actionable maintenance management of a critical system (Monte) in a timely, economical, and risk-controlled fashion.

The Data, Metrics, and Models

We collect a great deal of empirical maintenance data and have found that we can predict exceptionally well the demand for maintenance.

For example, maintenance effort is needed when users report defects from operating the system. Even though we cannot know exactly what these defects are or when they will be reported, they have predictable characteristics we can model quite well stochastically.

We use the defect-tracking system Bugzilla to manage defect reports and determine key maintenance information such as the code size over time, rate of defect discovery, time between defect discoveries, and effort required to repair defects. We have over 12 years of such defect data, enabling us to confidently model our maintenance process and generate useful metrics. Below, we discuss some examples.

Defect Discovery Rate and Code Size

To understand the demand for maintenance, we need to estimate how many defects we may have to address within a given time period.

We also need to monitor the overall quality of the system to ensure the system is maintained within acceptable defect risk tolerance. Figure 1 shows that the cumulative number of defects reported over time is nearly linear ($R^2 > 0.99$). So too is the cumulative code size over time.

Empirically, the rate of defect reporting is about 0.51 defects per day. Taking the ratio of the bug-reporting rate and the code size growth of 104 LSLOC (logical source LOC) per day gives an estimate of the mean defect density: $(0.51 \text{ defects per day}) / (104 \text{ LSLOC per day}) = 0.0049 \text{ defects per LSLOC}$. The defect density gives some insight into defect risk and system quality.

Time between Defect Discoveries

While the system is in operation, defects are continually being discovered, and we continually need to resolve them. To meet this demand for maintenance within acceptable

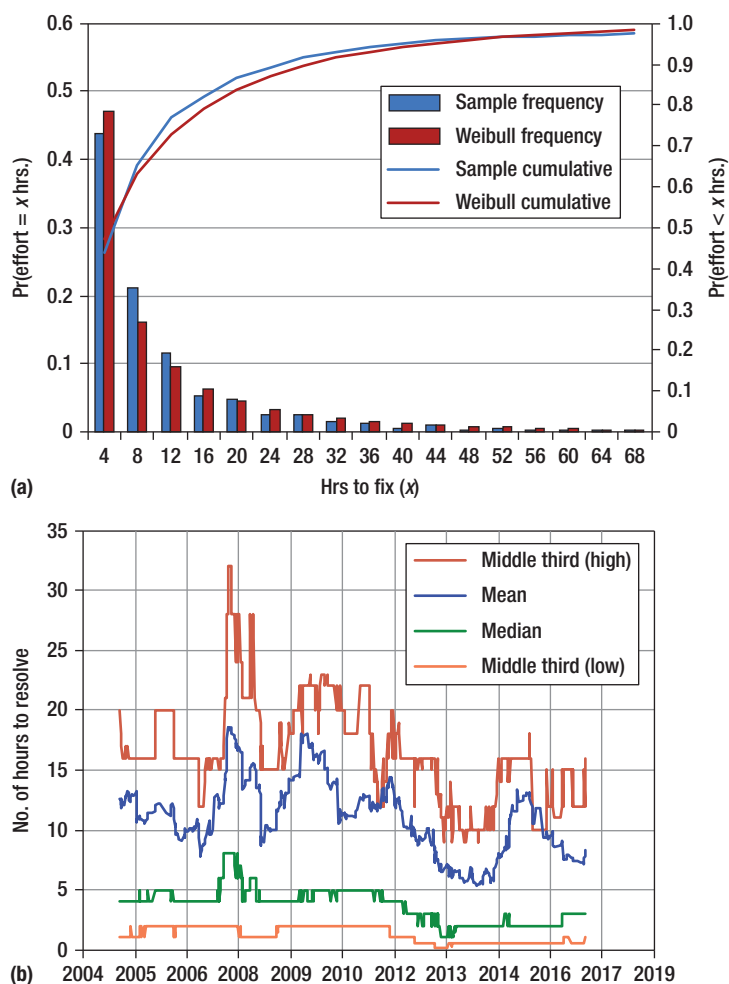


FIGURE 2. Fixing bugs in Monte. (a) The time to fix the bugs. (b) The evolution of the time to fix the bugs (trailing 101 bugs).

defect risk tolerance, we need to understand how often defects occur. Defects are discovered randomly, often in clusters at essentially the same time. The defect discovery rate of 0.51 defects per day indicates that a defect is discovered with a frequency of about two days. This rate quantifies the number of defects over time, but not the amount of time between defect discoveries. By looking at the

empirical distribution of times between when defects are reported, we find the median time between defect discovery for Monte is about 0.59 days.

Effort Required to Repair Defects

There will be an ongoing demand for maintenance with issues arising at any time; therefore we cannot manage maintenance effort

analogously to a schedulable software development effort. Failure to meet maintenance demand carries significant risk to the continued successful operation of the system and hence to the success of the missions that depend on the system. When defects occur, we need to know how long it will take to resolve them. For this, we track the effort required to fix each defect, and we find the effort required to repair a defect is modeled well as a Weibull distribution with parameters as indicated in Figure 2.

We track the moving tendencies (e.g., moving average) to better understand the variability and how stable and robust the estimates of the model parameters are.

The Analytics and Actions

MDN has models to determine defect rates and to inform our users about the reliability of a release and how that reliability will improve through a cooperative process of testing, reporting problems, and releasing repairs quickly. In addition, we use these models to ensure that senior managers are informed, both in principle and empirically, about the inherent risks in software maintenance and how schedules and budgets must accommodate this risk. Some fundamental actions and examples of analytics we use for them are discussed below.

Reporting System Defect Risk

Historically, we know that 7% of Monte failures are critical. This is based upon the user categorization of defects in the Bugzilla database. We try to keep the expected number of failures low and be prepared to respond quickly to the rare critical ones. However, mission failures (like

most failures) are generally the result of many small failures. By removing the small failures in a timely manner and ensuring that defect risk for the current release is within tolerance, we enhance the odds of success.

To determine these odds and the confidence that we are maintaining the system within risk tolerance, we must know how likely a defect will be encountered for a given release of the system.

The defect density from reported defects is not enough. We must estimate the number of undiscovered defects present in the system. By determining this distribution, we can estimate the current level of defect risk. By calibrating a Poisson distribution and a finite exponential discovery model, we can estimate the actual number of defects in the system at any given time.¹ We calibrate this model with inputs from the models discussed earlier.

For example, in the current release, we use the reported defect density in Monte of 0.0049 defects per SLOC, where 2,571 defects have been reported in the current code base of 744,295 LSLOC. We find that the distribution for the number of defects remaining in this release of the system is a Poisson distribution with a mean equal to $B = 744,295 \times 0.0049 = 2,571$. We find the mean defect discovery rate to be $D = 0.51$ defects per day. Thus, the defect “decay rate” given by $\alpha = D/B$ is $0.51/2,571 = 0.000474$. Using these estimated parameters, we can provide defect risk and system quality assessments for the current operational period.

Defect Repair Budget

Budgets are a basic control mechanism to ensure that effort will be directed to particular jobs. A basic

job that needs to be performed on Monte is the repair of reported defects. Determining how big this budget should be requires estimating the number of defects that will be reported in a year and how many hours, on average, each requires.

Using the linear defect-reporting model discussed previously for Monte, the expected annual number of reports is $0.51 \text{ defects per day} \times 365 \text{ days} = 186 \text{ defects}$. From the defect repair effort model, the mean repair time is 10.62 hours per defect. Thus, we should expect to budget $186 \text{ reports} \times 10.62 \text{ hours per report} = 1,972 \text{ hours}$ for maintenance. This, however, is a very crude estimate and does not account for the large variability in the number of reports and effort to fix the bugs. If we under-budget, we risk a shortfall in staff allocation and may find ourselves unable to address critical defects in a timely manner.

We can do much better by noting that the error in the number of reports estimated will be normally distributed. Combining this with Monte Carlo simulation and using the effort-to-fix distribution (Weibull) to get the distribution of the expected effort, we compute the 80th percentile (or other agreed-on risk tolerance) for the effort and use this for the budget estimate.

Allocating Effort for Maturing New Releases

The Monte software is in a state of sustained capability growth. Approximately 36,000 LSLOC of C++/Python code are added annually. With each growth in capability comes a new set of defects: about 186 per year. Using data from legacy defect discovery, as well as our models for discovery of defects in Monte,

we know that the half-life of a bug is about 0.75 years. This information, along with our knowledge of defect density, allows flight projects to understand how long it will take to address the inevitable new defects introduced.

Maintaining Appropriate Maintenance Staffing Levels

Staffing for maintenance of a critical software system requires more than measuring the number of staff hours needed for defect repair each year. Defects are discovered randomly, and the process to repair them is a random process.

Not being able to address a new issue allows the many small failures to accumulate. Our ability to address issues depends upon the staff available. Staffing needs to be adequate to ensure timely resolution of issues. Having sufficient staff on hand to continually address defects is risk mitigation. However, there will be times when the staff is not fully utilized, but we still must pay them. What is the smallest staff that mitigates our risk?

Using the current empirical distribution of defect reports, we can simulate the number of unaddressed defects over time with a fixed staff of developers. (See “Staffing Strategies for Maintenance of Critical Software Systems at the Jet Propulsion Laboratory”² for details on the simulation.) In this simulation, we have found that defect repair teams need a “bench” that is deeper than just the number of individuals required to keep up with the average rate of discovery and repair. Without this bench of developers who can step in to address a critical defect, the queue of defects awaiting attention can grow to a point where it

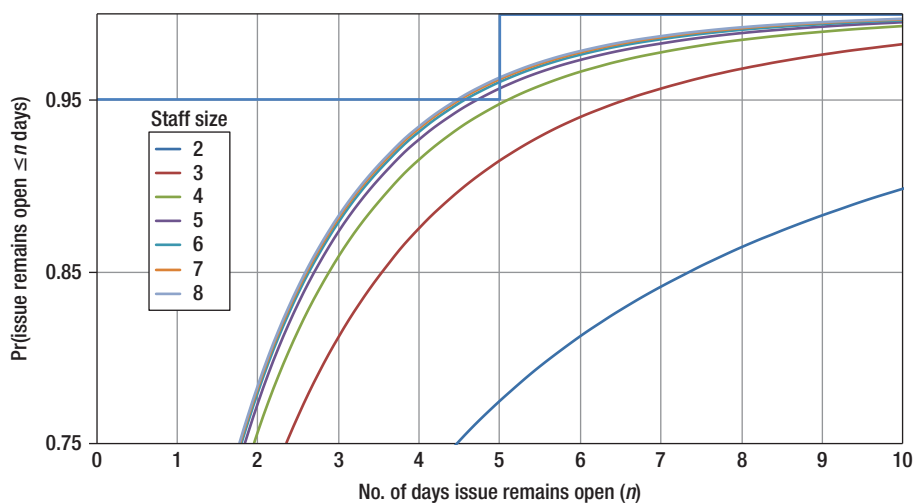


FIGURE 3. Simulated distributions of open-issue time for different staffing levels.

may require more than a month to clear a backlog of defect reports. Increasing staff can be viewed as mitigating the risk that a critical defect will go unaddressed due to the unfortunate discovery of several defects in a short time.


Using data models for defect discovery and repair, we can perform Monte Carlo simulations to determine the percentage of defects that can be repaired within a given number of days as a function of staff size. For each staff size, N , there is a new distribution for the likelihood that a defect can be addressed in n days. Using these distributions, you can determine a risk-appropriate staffing size.

For Monte, we have performed this simulation using the data models discussed previously as inputs, with the following results illustrated in Figure 3. We find that, for example, to get 95% of the issues resolved in five days, we need a staff of five

developers. Adding more staff would not improve this much, so this is a reasonable economic compromise.

Models and analytics are never perfect but are good enough to address our needs. Our models and analytics are developed and defensible on basic principles and empirical data, but more important, they are validated by their utility in managing the ongoing maintenance of our critical systems. We rely on them to take decisive action on reporting risk and determining budgets, schedules, and appropriate staffing levels.

Arguably, our most important result is that we could use our maintenance models and analytics to make a defensible business case to our sponsor that we needed more development funding to ensure we had a deep-enough bench to achieve an acceptable repair response time for

missions. Prior to this, we have had to rely on intuition and experience. The argument “If we don’t have this many people, things will be bad” was increasingly met with skepticism and budgets that endangered our ability to meet institutional needs. After presenting our business case to our sponsor, the sponsor is now keenly aware of the risk of low staffing, and we have been given funding to support the development of capabilities at a level sufficient to maintain the staff at a healthy level. 

References

1. W. Taber and D. Port, “Empirical and Face Validity of Software Maintenance Defect Models Used at the Jet Propulsion Laboratory,” *Proc. 8th ACM/IEEE Int’l Symp. Empirical Software Eng. and Measurement (ESEM 14)*, 2014, article 7.
2. W. Taber and D. Port, “Staffing Strategies for Maintenance of Critical

Software Systems at the Jet Propulsion Laboratory,” *Proc. 10th ACM/IEEE Int’l Symp. Empirical Software Eng. and Measurement (ESEM 16)*, 2016, article 49.

myCS

Read your subscriptions through the myCS publications portal at

<http://mycs.computer.org>

ABOUT THE AUTHORS



DAN PORT is an associate professor in the Department of Information Technology Management at the University of Hawaii at Manoa. His primary research area is empirical value-based software engineering; he specializes in software system assurance, strategic methods for development, and software and systems engineering education. Port received a PhD in applied mathematics from MIT. He’s a member of IEEE, Beta Gamma Sigma, and the International Software Engineering Research Network. Contact him at dport@hawaii.edu.



BILL TABER is the technical group supervisor of the Mission Design and Navigation Software Group of the Mission Design and Navigation section at the Jet Propulsion Laboratory. His research interests include software reliability and mathematical modeling of the software development process. Taber received a PhD in mathematics from the University of Illinois Urbana-Champaign. He’s a member of ACM and Sigma Xi. Contact him at william.l.taber@jpl.nasa.gov.

IEEE  computer society

Looking for the BEST Tech Job for You?

Come to the **Computer Society Jobs Board** to meet the best employers in the industry—Apple, Google, Intel, NSA, Cisco, US Army Research, Oracle, Juniper...

Take advantage of the special resources for job seekers—job alerts, career advice, webinars, templates, and resumes viewed by top employers.

www.computer.org/jobs

