Staffing Strategies for Maintenance of Critical Software Systems at the Jet Propulsion Laboratory

William Taber
California Institute of Technology
Jet Propulsion Laboratory
Mission Design and Navigation
Software Group
William.L.Taber@jpl.nasa.gov

Dan Port
University of Hawaii
Shidler College of Business
Information Technology Management
Telephone number, incl. country code
dport@hawaii.edu

ABSTRACT

Context: The Mission Design and Navigation Software Group at the Jet Propulsion Laboratory (JPL) maintains mission critical software systems. We have good empirical data and models for maintenance demand—when defects will occur, how many and how severe they will be, and how much effort is needed to address them. However, determining the level of staffing needed to address maintenance issues is an ongoing challenge and is often done ad-hoc. There are two common strategies are (1) reactive – add/remove staff as needed to respond to maintenance issues, and (2) capacitive – retain a given staff size to address issues as they occur, proactively address issues and prevent defects.
Goal: To use our empirical models for maintenance demand to address the issue of staffing from a risk perspective.
Method: We develop a staffing model that allows us to simulate large numbers of maintenance histories. From these histories we examine the risks posed to the institution as a function of the staffing available to address issues as they arise.
Results: We find that the model developed matches our intuition. There is a "sweet spot" in staffing levels that allows issues to be addressed in a timely fashion. Below that level the institution experiences substantial risk; staffing above that level does little to improve the institutions risk exposure.
Conclusion: The models developed provide tools that, for the first time, allow us to quantitatively discuss the level of staffing needed to ensure that we can meet the time constrained demands for maintenance on mission critical systems and thereby determine staffing budgets that ensure mission success.

Categories and Subject Descriptors
D.2.4 Software/Program Verification: Reliability

General Terms
Management, Measurement, Performance, Design, Economics, Reliability, Experimentation.

Keywords
software defect model, software maintenance, software reliability

ESEM '16, September 08-09, 2016, Ciudad Real, Spain
© 2016 ACM. ISBN 978-1-4503-4427-2/16/09…$15.00
http://dx.doi.org/10.1145/2961111.2962640

1. INTRODUCTION

The software used for NASA's deep space missions is critical to the success of those missions. For many of these missions, defects encountered and not repaired in a timely manner not only inhibit operations, but may also lead to the loss of a billion-dollar mission. Despite our many years of experience focusing on quality development and control, achieving zero defects in a delivered system has evaded us. We accept that defects in software are not just a possibility – they are a certainty. As a result, for every system in operation we must plan for maintenance effort to find and repair these inevitable defects. Given our limited budgets and staffing resources, and often a time critical need for repair, maintenance effort should not be planned ad hoc. It is simply too risky to have a critical system non-operational for an unknown period of time. Many tough experiences have shown us that poor planning for maintenance increases risk of mission failure.

Good maintenance planning must address questions of 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. The JPL Mission Design and Navigation Software Group has very good models for predicting 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 that schedules and budgets must accommodate the defect decay process to reach a desired level of reliability in operation of these systems.

In spite of having high confidence in these models, we grapple with the troublesome issue of determining an appropriate maintenance staff level to meet our expected maintenance effort. We view this as a supply-chain problem of balancing the costs of "holding an inventory" of experienced maintenance staff who can address maintenance needs on a continual basis with the "ordering costs" of obtaining maintenance personal (e.g. identification, training, etc.) to address maintenance requests as they occur or "demanded" by projects that will pay for maintenance effort. This is driven primarily by the ever present maintenance risk that the longer a maintenance issue is unaddressed, the greater the risk that the system will become non-operational or be exposed to a catastrophic defect.

2. MAINTENANCE DEMAND FOR TWO CRITICAL SYSTEMS

The JPL Mission Design and Navigation Software Group has two critical systems that have been operation over the last two decades

– a legacy navigation system and its replacement, Monte[2]. The legacy system is composed of over one million logical lines of heritage FORTRAN code. The initial development of this system began in the late 1960s and continued until approximately 2006. During the period from 1999 until the present, an effort was undertaken to replace the legacy navigation system with a new system, Monte, to be developed in modern languages using modern software development methodologies. As a result, in early 2006, the legacy system became a static (final release) system. Defects in the system have been repaired, but aside from this no new code has been introduced into the system. In addition, until 2012, the software has been in regular daily use by a nearly constant population of users during this time.

The adoption of Monte by flight projects was a slow process and required continuous management pressure. Flight projects are reluctant to accept new software. They readily accept bug fixes and small additions, but entirely new systems are a major challenge. During the period in question the navigation operations staff was very stable and nearly all were employed full time on flight projects: Cassini, Dawn, Deep Impact, Genesis, Hayabusa, MER, MRO, Odyssey, Spitzer, Stardust. The user base for Monte was initially a different population of users. These were the early adopters and for the most part the new people who "grew up" with Monte and never learned the legacy tools. The first mission to operate using Monte was Phoenix which successfully landed near the Martian north pole in May of 2008. Following this successful first mission use, all flight projects gradually adopted the new system. Monte has now successfully replaced the legacy product on all flight projects with the final transition by Cassini in 2012.

Software maintenance is the ongoing activity performed on a system post-delivery ostensibly to keep a system in operation. It differs from pre-delivery development in a number of ways in so far as how defects are discovered and removed and how requests for capability enhancements are handled. The exact maintenance needs are uncertain. However, what is certain is that maintenance issues will arise – software will fail and new or revised capabilities will be needed while the system is used.

We have been collecting empirical maintenance data since 2002. In that time, we have received over 4,000 service requests (bug reports and requests for new features. Using this large repository of data, we and have found we can predict exceptionally well the demand for maintenance [11]. 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, Figures 1 and 2 illustrate that for both the legacy system and Monte, the number of defect reports we expect is quite predictable.

The main question is, what will you do about it? From nearly two decades of experience here is what we have observed in regard to software maintenance:

- We know that 7% of failures are critical.
    - This is based upon the user categorization of defects. It's part of the *bugzilla* database. Critical means that the reporter can't do his/her job due to the defect.
- The environment in which software operates is changing:
    - The computer you are using today is not the one you will be using in 3 years.
        - The OS will be different when you get your new computer
        - Mission critical software will need to be ported to the new system
    - OS patches and upgrades are not entirely your choice. Due to security vulnerabilities or desire/need to employ other software on your systems, your system operating system (OS) will be upgraded. An OS upgrade requires porting of your essential software. Moreover, a new OS is likely to spawn new defects in working software.
- Your mission is evolving.
    - You need to interface with other software that did not exist when your software was created
    - Your requirements were incomplete. You will want the software to do things it didn't do when it was released.

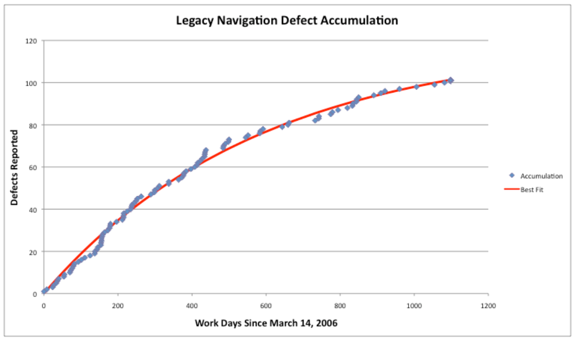


Figure 1. Model of accumulated defect history for legacy navigation software.

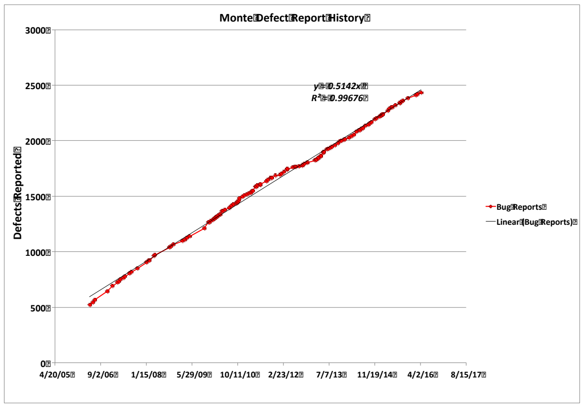


Figure 2. Monte Defect Accumulation

3. MAINTENANCE STAFFING MANAGEMENT

Recall the main question we are trying to address is how to determine an appropriate maintenance staff level to maintain an institution's unique essential software e.g. Monte.

This is poses a serious challenge for several reasons:

- 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 systems and hence the success of the missions that depend on the systems.
- Funding for maintenance effort is complex and from multiple sources - coming from projects that request maintenance or from institutional "operating and programmatic" funds that are limited and in high demand.
- It takes significant time and cost to hire and train maintenance staff to address new maintenance issues.
- Experienced staff are generally not available on demand when issues arise. They are working on other tasks.
- Maintenance staff may be "idle" when there are lulls in maintenance demand, but costs (salary) for them continues.
- There is an "opportunity cost" in holding experienced staff in reserve when they could provide value on another task.

Our experience in ad-hoc management of staffing (i.e. guessing at our staffing needs and hiring only in response to demand) has proven problematic in grappling with the above challenges. In particular, we have frequently found ourselves in critical need of maintenance effort with a very limited supply staff available or that can be hired to meet our maintenance demands. This usually has to do with a new project that wants some new set of features in a short time span. But since there is no slack in the system, we have been unable to respond. Hiring a new person isn't feasible since the work needed will only last a few months.

For much the same reasons listed above it is not feasible, and also quite risky, for us to experiment with different staffing levels to determine a sensible strategy. As a result, we are building a justifiable and defendable model driven by our systems defect models to justify resources (i.e. funding and staff) for maintaining an appropriate maintenance staff level.

We formulate this model in terms of a supply and demand for maintenance and compare strategies for meeting maintenance demands. Here is an overview of this with details in subsequent sections. Our defect models show us over time stochastically:

- the number of maintenance issues

- when they will occur

- how long it will take to resolve them

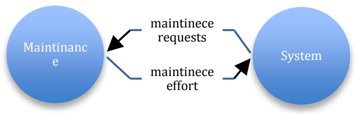
Maintenance effort is the "supply" to meet the "demand" from maintenance issues that arise from defect occurrences. Note that for us a defect may be a missing or desired capability for the system and not necessarily an error.

The more the demand is not met (i.e. overdue or unaddressed maintenance issues) the greater the system operational risk (i.e. risk of faults + risk of errors + risk of inadequate capability for the mission). Such risk is a significant component of mission risk as missions critically depend on the continual operation of the systems and their fitness for their use on their particular mission.

Business strategy involves leveraging the core competencies of the organization to achieve a defined high-level goal or objective. For us there are two basic strategies maintenance effort can be supplied:

1. Reactive Maintenance (ad-hoc or "pure pull")

When the risk for not addressing outstanding maintenance issues is over a certain tolerance (i.e. lots of little issues or perhaps one critical one) hire staff to address the issues (based on expected effort).



This strategy has the following benefits:

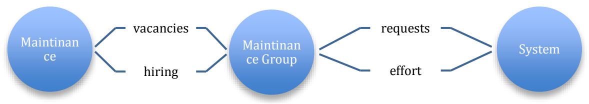
- Agility – adjusts to demand needed, no forecasting
- Custom configured – specialized to particular needs
- High utilization – staff will never be idle
- Low institutional operating cost – effort costs are funded by maintenance requestors (projects)

This strategy has the following detriments:

- High risk of shortfalls – inability to hire sufficient staff to meet demand
- Unaddressed issue risk – long times to hire staff cause delays in addressing maintenance requests
- High staffing cost – use of non-dedicated staff, training costs, emergency staffing to deal with critical issues
- Budget risk – projects do not budget for maintenance contingency and high variability in staffing needs pose the risk that funds will not be available to cover maintenance demand

2. Capacity Maintenance (proactive or "push-pull")

Have an "inventory" of staff on hand to address maintenance issues as they arise. We add and remove staff in anticipation of need based on demand forecasts.



This strategy has the following benefits:

- Low risk of shortfalls – likely to have sufficient staff to meet demand as it occurs
- Low unaddressed issue risk –maintenance requests are addressed as they occur
- Stable staffing cost – low variability is staffing level
- Low budget risk – the majority of staff costs come from operating and programmatic funds that projects budget for in advance rather than unbudgeted maintenance requests

This strategy has the following detriments:

- Risk of overage– staff level is driven by forecasted demand. To maintain a high-level of service (i.e. meet demand) a certain amount of "safety" staff will need to be on hand to hedge variable demand
- Utilization costs – staff may be idle during maintenance demand lulls.
- High operating cost – staff salaries must come from highly constrained operating and programmatic funds.

The factors involved in comparing these strategies relative to institutional goals are:

Underage - cost of not having maintenance capability to address issues (in terms of overdue maintenance issues risk that could have been avoided)

Overage - cost of having maintenance capability available that is not used to address issues

Supply cost - cost of acquiring maintenance staff

Lead time - time it takes to supply maintenance effort (we can assume this is stochastic)

Maintenance demand - based on outstanding issues (stochastic) over some period of time. This demand period can be somewhat arbitrarily chosen to best suit the data collected or organizational management. Since demand occurs over time, we are just describing the demand over a window.

Maintenance risk - based on outstanding issues (stochastic) over some period of time

There are risk and cost advantages and disadvantages in terms of the "total cost" (i.e. overage + underage + actual costs over a given maintenance demand period) using the above factors for each of the strategies.

We can tune the capacity strategy (2) by finding the amount of maintenance effort inventory that meets the institutional risk posture.

Given the above, the questions we wish to use our model to address are:

1) Are there significant differences in total cost for the two strategies?

2) What level of risk reduction associated with the capacity model is sufficient to justify full time employment (FTE) in terms of salary commitment from restricted operating and programmatic funds?

3) It is possible to combine these two strategies for a hybrid that may be better in terms of total cost and organizational issues (e.g. constraints or incentives for commitment of operating and programmatic funds)?

4. MAINTENANCE SUPPLY

Maintenance requires a staff capable of modifying the source, the build environment, and installing and training people in the upgrade. However, the level of staffing needed for this task is not well known. It is almost certainly greater than most organizations, ourselves included, believe it to be.

Let us consider the risks:

- Critical failure
  - The repair cost is not the issue. The organization stands to have severe losses as a result of the failure if it cannot be repaired in a timely fashion. Hence time to getting to a repair becomes important here.
  - A year's salary of a having a developer sit on his/her hands would surely be less costly than the loss from a critical failure
  - It is important not to overstate likelihood of a critical failure. Critical operations tend to have test periods prior to putting the software into live use. However, given the large magnitude of loss, the risk is still significant even when the likelihood is small.
- Environmentally induced defects
  - An OS change gives rise to a new defect. These defects expose a change in the computing environment that may be more wide spread within your systems. They are not as problematic, as you can postpone the upgrade while your staff addresses the change.
  - If you have inadequate staff, new development may be deferred.
  - You can staff up—but the time to staff up is not instant and measured in ½ years.
- New requirements
  - New interfaces
  - Extending capability
  - This is a matter of opportunity. You might choose not to take advantage of the opportunity; you let the business go elsewhere and hence an opportunity loss if you don't take it. This is a business decision with a potentially large long term impact and this it is a risk.
  -

In addition to constructing an accumulation model, a decay model is constructed for the observed defect history. This model enables us to forecast how much maintenance can be done and how long it takes to address issues. An example model of this us shown in Figure 3.

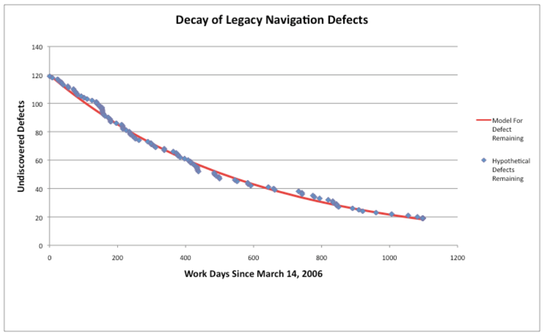


Figure 3. Model for the decay of defects in Legacy Navigation software.

5. DEMAND-SUPPLY MODEL

With an understanding of the demand for maintenance and our ability supply maintenance effort we now formulate and analyze a model of the situation currently facing the software development team for the Mission Design and Navigation Software group at the Jet Propulsion Laboratory. Using historical data for over 2500 defect reports since 2004, we have on hand empirical models for the time between defect discoveries, the priority of defects, and the effort required to resolve a defect.

Using Monte Carlo sampling from these empirical models we simulate a history of issues for a 1000-day period. This history is then processed by staffs of various sizes (2 through 8 individual staff members). From these simulated histories statistics are gathered on the length of time issues remain unresolved, the number of issues assigned to staff members as a function of time.

By simulating 1000s of such histories we have determined empirically the ability of a team to keep issues from leading to catastrophic consequences.

Issues: Issues are defect reports. They have a severity (major, normal, minor), a creation time, an estimated expense (the number of days required to resolve the issue based on historical norms), an expense (the actual number of days needed to address the issue if worked on without interruption), a due date, and a cost associated with the issue if it is not resolved by its due date.

Staff Members: A staff member models a member of a development team. A staff member has areas of expertise, and ability to learn (and forget) new domains, and a list of prioritized issues to be addressed that grows and shrinks as issues arrive and are resolved.

Product Manager: A product manager has a team of staff members. The product manager handles the assignment of issues to team members to ensure that work is balanced across the team so that backlogs of issues remain small. To do this, when a new issue arrives the manager polls each staff member on the team to determine the estimated cost that would be incurred if the issues were assigned to that staff member. The issue is assigned to that staff member who can work on the issue with the least overall resulting cost (costs include other issues that might be delayed due to work on the issue).

Minimum Team Size: Issues are produced at an average rate of r issues per day. A staff member requires, on average, E days to resolve an issue. Thus the number of issues a staff member can resolve per day is $1/E$. Staff members work independently so that k staff members can resolve k/E issues per day. In order for the unresolved issues to not produce an every growing backlog, the number of issues a team can resolve per day must be greater than the number of issues generated per day: $k/E > r$. Thus the number of staff members, k, must be at least as large as rE. ($k > rE$).

Issue Details: Issues arrive at random in accordance with some probability distribution of the times between issues. We have used the historical record of defect reports as an empirical distribution for the time between issue creation. This model is shown in Figure 4.

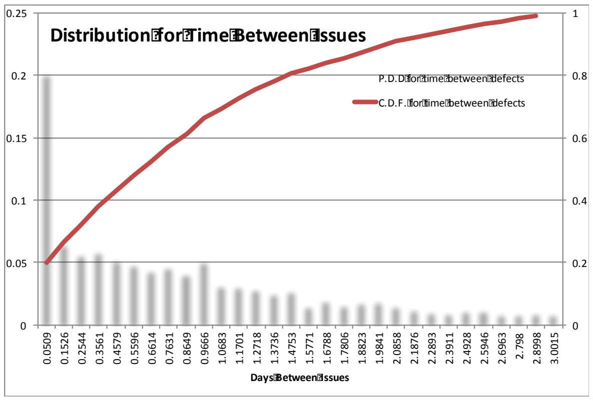


Figure 4. Time between issue creation

Issues have 3 possible levels of severity: major, normal, and minor. Using historical data, we have modeled the relative frequencies of these severities as 7%, 78% and 15% respectively. The effort required to resolve an issue, if worked on without interruption, follows a Weibull distribution with mean 10.9 hours and standard deviation of 13.1 hours.

Issues are associated with different portions of the software system. The relative frequency the various categories of issues are shown in Figure 5.

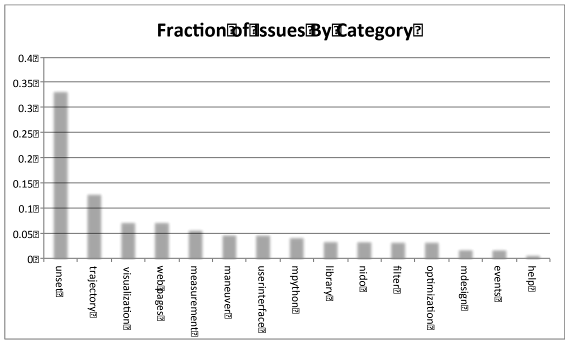


Figure 5. Occurrence of issues

Issue Histories: An issue history is a random sequence of issues, ordered by creation date that span 1000 days. The issue history is provided first to a Product Manager with a development team of 2 staff members, then to a Product Manager with a team of 3 staff members and so up to a team size of 8 staff members. Data on the average queue of issues across the team are recorded, the time issues remain open, as well as other data relating to the resolution of issues. This set of activities is called a *simulated history*. The relationship between staff size and the risk of not resolving an issue on time is determined by aggregating 1000s of simulated histories and examining the likelihood of a critical issue not being resolved in a timely manner.

6. Results

Initially, it was thought that we would look at the problem of maintaining a system by starting with a small staff and then adding staff as a backlog of issues grew. However, one quickly sees that if one is shorthanded, issues become backlogged almost immediately so that hiring a new staff member becomes inevitable and the initial transient backlog fades back to the history of having started with this number of experienced developers.

Thus the risk question associated with staff size is one of determining how long you can wait for an issue to be resolved by a team member. Does an issue need to be examined and resolved in a day or two? Can you wait two weeks? Two months?

It also quickly became clear that picking due dates could not be justified on the basis of experience. Our historical data does not contain information about "need by" dates. In addition, we don't have data on the cost of issues that are not resolved in a timely fashion. We only know that the cost can be very large: the loss of a mission, the loss of future missions, the loss of enabling infrastructure assets, etc. The losses in these situations is 100s of millions of dollars. They are the magnitude 8 events that shake an organization such as JPL. These issues are the outliers. They lie in the "tails" of the cost distribution. They are the issues that better not slip through the cracks. We don't know when they will arise. They are rare, but are clearly not one-in-a-million kind of events. They seem to arise once every decade or so. Often they are not obviously urgent but just some anomaly that shows up in some part of the system that is not easily explained. Nevertheless, they tend to require prompt attention. For an issue to receive prompt attention, some staff member needs to be able to address it as soon as it arrives i.e. some staff member needs to have an empty queue of pending issues.

Since we are interested is discussing maintenance from a risk perspective, we choose to model the distributions of *queue depths* and *resolution times* as a function if staff size. Queue depth is the number of issues a developer has waiting to be worked on at any time. Resolution time is the time it takes for an issue to be resolved (worked and delivered to operations) once it has been identified. The chart in Figure 5 shows the percentile queue depths of queued of issues facing a staff member when a new issue arrives for 80th, 90th, 95th and 99th percentiles based upon the staff size. Depending upon your risk posture you can determine the staff size you need to address a percentage of issues when they arrive.

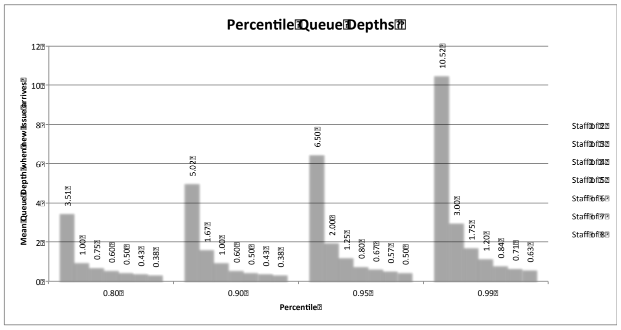


Figure 6. Percentile queue depths for given staff levels

As is evident from the chart, staff sizes of 2 and 3 are inadequate to address even 80% of issues promptly (if we define prompt to mean that there is no more than one issue present in a developer's queue when a new issue is assigned to the developer). The minimum staff sizes to address 80% of issues promptly is 4, for 90% 5, for 95% 5 and for 99% 6.

A similar chart in Figure 6 shows the staff size needed to resolve issues within a given time span.

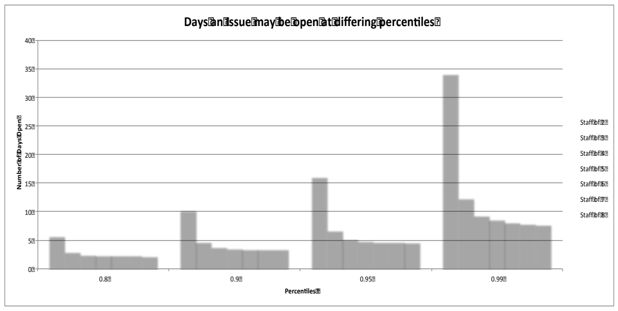


Figure 7. Time to resolve issues given staff level

7. CONCLUSION

The models discussed here are being used in the Navigation Software Group at the Jet Propulsion Laboratory to provide insight into determining a strategic maintenance staffing level. The institution has many demands for programmatic funds. Committing a portion of these funds to cover staff salaries solely for maintenance, especially when the normative is to have individual projects pay for their maintenance requests, requires an informed discussion of risks.

Prior to the development of these models we could only point to our challenges in obtaining staff and inability to address possibly critical maintenance issues as reasons that our ad-hoc maintenance staffing is risky and inefficient. However, with these models we are now able to provide the institution with a more informed business case to identify the level of resources they need to devote to maintaining reliable and continuous operation of our software to help ensure mission success.

Without a strong, defensible business case, the budget for sustaining a maintenance effort was largely a matter of opinion. There was a natural tendency for the software development team to request more than was needed. And there was a natural tendency for the sponsoring programs to regard the development teams asserted need as exaggeration and provide too little funding. With the advent of these models, both the program and the development groups can now reach a comfortable, informed, agreement on the resources needed to sustain the institutional investment in our mission critical navigation and trajectory design software. This does not mean that budgetary pressures have been removed, but both sides of the budget issue can now address the issue from the perspective of a *quantified* level of risk and cost and then debate can focus more on acceptable risk tolerance rather than speculation on need and (usually wildly optimistic and over-confident) ability to address need.

The models presented are selected on principles and backed by our empirical defect models. There is certainly error in these models and in the necessarily simplified view of maintenance demand and supply we have formulated. However, we do not require highly accurate results, rather we need to have insight into the general characteristics and impacts of the two basic maintenance staffing strategies. Our results are not greatly sensitive to perturbations in the factors and so we can have a reasonable degree of confidence in them despite the inaccuracies of our models and estimates of their parameters.

8. ACKNOWLEDGMENTS

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

9. REFERENCES

[1] Casella G, Berger R. *Statistical Inference 2nd Edition*. Duxbury Press. 511 Forest Lodge Road, Pacific Grove, CA 93950, USA

[2] Evans S., et al. "Monte: The Next Generation of Mission Design & Navigation Software." The 6th International Conference on Astrodynamics Tools and Techniques (ICATT) March 14-17 2015, *Proceedings* Darmstadt Germany

[3] Garrison Q. Kenny. "Estimating defects in commercial software during operational use." IEEE 1993.

[4] J.D. Musa. "Validity of Execution-Time Theory of Software Reliability." IEEE Transactions of Reliability, Vol R-28, 1979. pp181-191. J.D. Musa, K. Okumoto

[5] Jelinski, Z., and Moranda, P. "Software Reliability Research," *Statistical Computer Performance Evaluation,* Freiberger, W. Ed. Academic Press, New York, NY

[6] Kan S. *Metrics and Models in Software Quality Engineering 2nd Edition.* Addison-Wesley, Boston MA, USA

[7] Li, P., Mary Shaw, and Jim Herbsleb. "Selecting a defect prediction model for maintenance resource planning and software insurance." *EDSER-5 affiliated with ICSE* (2003): p32-37.

[8] Li, Paul Luo, et al. Empirical evaluation of defect projection models for widely-deployed production software systems. Vol. 29. No. 6. ACM, 2004.

[9] Neufelder, A M. *Ensuring Software Reliability.* Marcel Decker, Inc. 270 Madison Avenue, New York, NY 10016, USA

[10] Savage, L.J. *The Foundations of Statistics.* Dover Publications, Inc. 180 Varick Street, New York, NY 10014, USA

[11] Taber W., Port D., "Empirical and Face Validity of Software Maintenance Defect Models Used at the Jet Propulsion Laboratory" ESEM '14 Proceedings of the 8th ACM/IEE International Symposium on Empirical Software Engineering and Measurement. Article No. 7.