Software Maintenance: Debunking the Myths

Donald J. Reifer, Reifer Consultants, Inc.
Jill Ann Allen, U. S. Army
Brian Fersch, U. S. Air Force
Barbara Hitchings, SAIC
James Judy, Office of Deputy Assistant Secretary of the Army (Cost and Economics) and
Wilson Rosa, U.S. Air Force Cost Analysis Agency

Abstract: This paper provides a snapshot of the findings, conclusions and recommendations of a year-long Joint U.S. Army and Air Force study on the topic of software maintenance. Using facts gathered during fact-finding, the paper focuses attention on dispelling myths about maintenance. While focused primarily on weapons systems, the conclusions offered seem generally applicable to any system, either military or commercial. The ultimate goal of our recommendations is to improve practices used for software maintenance in the field.

1. Introduction

During the past year, our study team has investigated in-depth how software maintenance was being conducted across the aircraft, missile and space domains [1]. The investigation has canvassed nearly forty Air Force and Army major weapons systems projects via questionnaires and interviews to develop conclusions and recommendations. As part of this effort, we tried to understand what tasks software maintenance personnel actually perform day-to-day as part of their jobs as part. Both large and small projects were canvassed across acquisition and software life cycle support shops. Projects both in and transitioning into maintenance were interviewed.

Based on the tasks identified during fact-finding, we next looked at how these work involved was estimated, budgeted and managed. We also looked at the metrics and cost estimating models used for gaining insight into the work progress. As the study's final thrust, we tried to gather hard data to bound the actual costs of maintenance and identify what the primary drivers were that influenced software productivity and product quality.

2. Study Approach

The study approach used to perform this study is pictured in Figure 1. We started by establishing "identifying better ways to estimate plan for software maintenance for weapons systems" as our primary goal for the effort. We next conducted an extensive literature search and a review of past study efforts to establish a baseline point of departure. Most of the publications we found during our investigations seemed dated and painted a distorted picture of the topic [2, 3, 4].

As our next step, we conducted fact-finding by interviewing key personnel involved in software life cycle support for aircraft and missile system projects at Redstone Arsenal in Huntsville, Alabama. We developed a questionnaire which was sent out prior to fact-finding to gather data and structure our interviews with project personnel. We next developed our initial findings, conclusions and recommendations using the facts, opinions and hard data that we gathered via these interviews. As our next step, we visited and interviewed personnel at Picatinny Arsenal and Fort Monmouth in New Jersey to confirm that our findings were Army-wide. To further validate our findings, we visited Air Force Material Command at Hanscom Air Force Base in Massachusetts. We further confirmed the validity of our findings there by interviewing eight Air Force programs currently in or transitioning into maintenance. Based on these visits and interviews, we believe that the initial results of our study present a valid picture of how life cycle support is currently being addressed by the Army and Air Force weapons systems communities.

Figure 1 - Study Approach

3. Study Findings

What we found tended to be counter to popular belief. For example, we were able to ascertain the work that maintenance centers performed was quite different than that perceived by taking the data gleaned from our questionnaires and interviews. As illustrated in Figure 2, the work involved much more than just software maintenance (i.e., the block release process). We found that about half of the work done within these projects did not involve block releases at all. Instead, it dealt with supporting oversight by program offices (acquisition management), performing independent test and analysis) and pursuing new software development. Equally interesting was the fact that those doing software maintenance tasks split their work almost equally between generating new releases and sustaining engineering activities (i.e., maintaining the infrastructure and test facilities required for the conduct of life cycle support). This is interesting because the general perception in the weapons system community is that most of the effort in software maintenance is focused almost entirely on generating new releases [5].

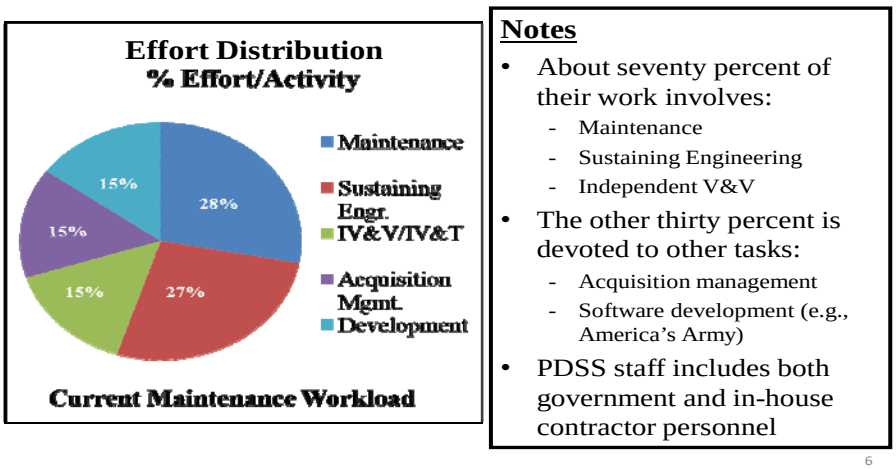

Figure 2 – Work Performed by Maintenance Centers

When we looked at who performed these tasks we found that work was primarily accomplished by government-led teams with contractor support. When done at government sites, the contractors resided in-house for the most part. In other cases, the team managed some prime contractor doing the work in their own facilities (i.e., provided oversight). In some cases, the government led teams worked directly with the prime contractor in an Independent Verification and Validation (IV&V) or Independent Verification and Test (IV&T) role. As part of their effort, they provided the prime contractor with recommendations for improvement based on their independent analysis and test results. In most cases, all of the teams functioned such that it was very difficult to determine who was wearing what badge. The well-managed maintenance teams were all directed on keeping system operational independent of whose badge employees wore.

As illustrated in Figure 3, we were also surprised when we looked at the contents of current releases in terms of work content. Folklore based on studies performed in the 1980s at UCLA led us to believe that this work for major releases was mostly enhancements or adaptive changes [6]. While true in most cases, the work was distributed 70/30 between enhancements and perfective changes and repairs that fix open trouble reports versus the 80/20 claimed in the literature. This is a significant finding because it says that more and more repairs are creeping into content. The root cause behind this phenomenon seems to be that most projects within life cycle software organizations were unable to fully address their repair backlog because of other demands and funds limitations. They fixed high priority problems and deferred the others to later releases.

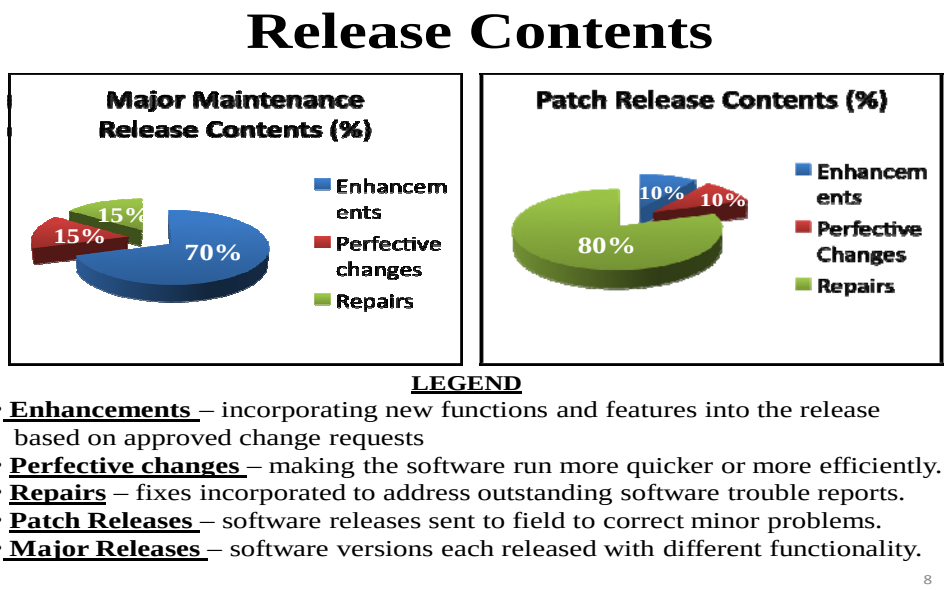

Figure 3 – Release Contents

When we look at sustaining engineering workloads we see that most of the workload revolves around making repairs aimed at keeping the system operational. As much as eighty percent of the work occupies the content of patch releases that are generated as a product of sustaining engineering activities. When we look deeper, we see most of these repairs respond to high priority trouble reports. The leakage from patch to maintenance releases seem to be caused by the extensive backlog. In other words, only a fraction of pending trouble reports tended to be worked at any given time. Those remaining open trouble reports represent a backlog that has not yet been incorporated into the pending release.

We also were able to glean certain important technical facts during fact-finding. For example, we saw that eighty percent of the problems came from twenty percent of the software modules. Based on this finding, it might be prudent to replace error-prone modules rather than repair them when funds for such activities could be found. Of course, this recommendation needs to take into account the possibility that new errors may be introduced as a result of the fix. It also must track the backlog to ensure that high priority fixes do not negatively impact deferred problems.

When we look at the technical work performed by the projects that we interviewed, we found that the vast majority of their activity being performed on both large and small projects in life cycle support revolves around testing and retesting. This is not surprising when you think about it when the projects involved are small and most of their effort is focused on maintaining their test infrastructure and generating patch releases for the field. However, bigger projects performed a great deal of testing as well. They tested at multiple levels using regression test baselines to revalidate that the release was fit for operational use. In addition, many of these larger projects had to perform series of operational tests as well to ensure that changes did not alter their fielded capabilities in ways that put soldiers at risk.

Unfortunately, we observed that many of those transitioning software to maintenance support make the job of testing in maintenance harder by failing to deliver a set of regression test cases for use in revalidating the software once changes are made. Delivery of such test cases was not included as part of the contract in most cases because it was not deemed a necessary expense. In addition to these testing and the traditional requirements, design and coding tasks, maintenance centers are often saddled with many support tasks which seem to be underfunded. Such tasks included those configuration management and quality assurance that are required to maintain system integrity and quality during maintenance. Other tasks like distribution management and user support that were needed to support field operations were often inadequately funded as well.

The ultimate goal of life cycle software support is to keep the system working operationally and support the mission. Without such support, it would be difficult for software life cycle support centers to accomplish this goal. Projects in maintenance seem to get this job done. They tend to be nimble and resourceful and do the best that they can with the resources that they are provided to get the job done to the best of their abilities.

4. Myths of Maintenance

There were ten major myths discovered as we conducted the study. For the most part, these myths were defined as findings that defied current perceptions and folklore relative to software life cycle support. We first explain the myth and then debunk it in the paragraphs that follow.

<u>Myth 1 -</u> most maintenance jobs were aimed at addressing new requirements.

Because the majority of maintenance projects that we examined were small, their primary goal was to close high priority trouble reports not address changes. Because these projects were funded LOE (Level of Effort), they could not handle the entire volume of work involved. Instead, they fixed problems based on priority and left the remaining as backlog. Over the years, this backlog grew and the focus of the maintenance center changed to reducing backlog. This turned out to be their main goal. While requirements played a major role in the mix, most of the fixes were aimed at correcting defects in these smaller projects, not implementing new features and functionality as in the larger projects. As such, based on the projects interviewed to date, we can therefore state for smaller projects that the software life cycle support workload is aimed at handling backlog, not satisfying new requirements.

Myth 2 – maintenance is funded based on requirements

This next myth focuses on funding misperceptions. In new developments, budgets are developed based on requirements. However, maintenance projects differ in that most are given budgets and told to get as much done as possible based on the funding level provided. This is especially true for small projects like many that we interviewed where funding is provided on a level of effort basis and all tasks including infrastructure and support had to be covered using this funding. The majority of projects (i.e., about eighty percent) that we interviewed during fact-finding had less than ten people assigned to them. For the most part, these projects had to rely on their staffs to do everything (i.e., they had no separate administrators or support groups to call upon for help). For the remaining twenty percent of the projects which were large, requirements governed content as generating block releases was the norm.

Myth 3 – maintenance schedules are based on user need dates.

Our third myth is logical when you think about it. On new jobs, you determine how much it will cost and how long it will take to get the job done based on requirements. The situation during maintenance though seems very different especially for smaller projects. Here, you are told to try to get as much done as possible by the next release. This release date is fixed. If you miss the date, the fix (change request, trouble report, infrastructure change, etc.) will be incorporated into the backup, not the current release. Essential capabilities are scheduled first and the rest comes whenever projects can get to it. Therefore, we can state that release date drives schedule, not need date. Hopefully, need and release are synchronized. But, this does not seem to be the case in all cases. Just as important, there are no penalties for missing dates if the fixes are not essential. Backlogs grow as a consequence and managing deferred changes becomes an issue.

Myth 4 – sustaining engineering effort is separately estimated and managed

This next myth focuses on how software life cycle support projects are estimated and budgeted. As we noted, maintenance projects seem to be estimated to address the maintenance release workload. Few have separate line items or budgets for sustaining engineering. Cost models that are used to predict costs for the most part do not estimate the full costs. For the most part, it seems that most Program Offices responsible for managing these projects assume that the infrastructure costs and other sustaining engineering tasks are accomplished as part of the mainline block release process. On a few projects, separate funds were provided to sustain the test facilities and integration labs. But, this was the exception rather than the norm. When it comes to sustaining engineering tasks, they use whatever mainline maintenance funds that are provided to the project. This conflict often results in less getting done than planned.

Myth 5 – IV&V uses separate processes, people and tools to assess the capability of the release to perform operationally

Our fifth myth focuses in on IV&V (Independent Verification & Validation) practices. The literature recommends that separate processes, people and tools be used to perform IV&V tasks [7]. The reason for this is simple. If the processes, people and tools are independent, so will be the results. The probability that errors introduced inadvertently would be caught increases along with the perception that biases would be eliminated using such processes. Unfortunately, projects interviewed did not have budgets that accommodated true independence when it came to processes, tools and people. Tools and people were for the most part shared because of lack of funds. Because facilities and equipment were limited, the IV&V team was also handicapped when it came to accessing the tactical equipment they needed to perform their tests. The good news was that processes were in some cases different and honest attempts were being made by the IV&V teams to act as honest brokers when it came to assuring the integrity and quality of the weapons system code.

Myth 6 – maintenance people are junior.

This sixth myth was a major surprise. The perception in industry is that software maintenance efforts are heavily populated with junior people. Most think that the best personnel go to development groups because that is where the glamour and action is concentrated. While this perception seems true to a degree, the levels of skills and experience we saw on maintenance projects countered this misconception. Because system in maintenance employ outdated languages (Ada, Jovial, etc.), architectures, equipment and operating environment, senior people were needed to maintain them. Junior people just did not have the skills, knowledge and experience required to get the job done within the timeframes required.

Myth 7 – motivating maintenance staff is hard.

Countering the previous myth, we have found that motivating senior people is easy, not hard. They thrive when provided interesting work, educational opportunities, good tools and a pleasing work environment. Like most professionals, they like to be pampered and praised and sheltered from bureaucratic overkill. In other words, give senior people a challenging job and the ability to excel and they will respond positively. Based on these findings, we believe that motivating senior people working maintenance is not as hard at all.

Myth 8 – process improvement addresses software maintenance

Our next myth addresses the topic of process improvement initiatives for maintenance. When you look at these initiatives carefully, you will find that both the ISO 9000 and CMMI process frameworks are primarily aimed at managing new software development work. While they provide structure for management of product releases, they do not seem to provide insight into the types of work performed on maintenance tasks. That is a shame because more than half the work in most software shops involves maintenance of existing systems.

Myth 9 – the only job maintenance centers do is maintenance

Our ninth myth highlights our finding that maintenance centers do much more than just software maintenance. As we have discussed throughout this paper, they do sustaining engineering, IV&V and acquisition management tasks as well. In addition, they may even get into the software development business. There is good news here as well. Because software life cycle support is performed primarily by government-led teams, maintenance centers have the flexibility to employ new paradigms when developing software. They also have the ability to use commercial best practices when they make sense. The high CMMI ratings of several of the software life cycle centers that we visited provide evidence that disciplined development practices can be and are employed to perform routine maintenance tasks.

Myth 10 – maintenance centers focus almost entirely on software maintenance.

This myth highlights the fact that most maintenance teams are called on to do much more than software. These primarily small teams typically perform systems engineering tasks, work contractual issues for the Program Office and fix both hardware and software. Their job is to make the system work operationally. Because of this, they have to be able to handle just about anything that comes their way in order to keep the system operational.

5. Overarching Issues

When asked about issues, not surprisingly those interviewed cited money issues as their primary concerns. These concerns ranged from shortfalls in funding to types of funds allocated (i.e., one year versus two year money). Type of money was raised as an important issue especially on pertinent on government projects where restrictions are imposed due to funding type. Budgets for sustaining engineering were noticeably inadequate. Perhaps, this was caused by shortfalls in estimating models that assume maintenance is the only work performed by these staffs [8].

In the area of shortfalls, lack of funds for tools, tactical hardware and test gear headed the list. In addition, getting sufficient funds to cover software license commitments seem to be a common theme as was the lack of funding for information assurance and quality improvement activities.

Because of its importance, many of those interviewed highlighted the need for improved budgets and timelines for transition and transfer activities. Most development projects seemed to rush transition. These projects neither adequately planned for transition nor allocated sufficient budgets for required transition and turnover activities. Maintenance shops suffered greatly because of this lack of attention. They were therefore often called upon during the first year of operations to perform heroic actions make the system work.

6. Study Recommendations

As part of this effort, we are developing a Handbook [9] aimed at helping those estimating and budgeting maintenance activities prepare more credible inputs (i.e., based on experience, not folklore and funds availability). The Handbook that we are developing could also be used to set funding expectations on the part of Program Offices and establish "should cost" estimates for use in confirming budgets. It could also serve as a resource for use in performing trade studies and for developing make/buy analyses and for life cycle cost analysis.

As a second task, we are developing recommendations for new metrics and estimating models to better bound maintenance costs and progress. Pursuit of these recommendations will be accomplished in such a manner that they could be cited by the Handbook. Based on a parallel analysis, the most popular cost models used for estimating software costs really do not fully bound the costs of all of the work that is performed by maintenance centers.

We have initiated dialog with the major software cost model vendors to address the issues involved in software maintenance estimation [10] [11] [12] [13]. We also plan to conduct workshops during the next year on the topic of maintenance to further engage the community.

In addition to fixing this problem, we intend to identify metrics during the next year for that maintenance centers could use to gain better visibility and control over their efforts. Metrics currently being suggested for maintenance have distinctive development flair. Development of new metrics and measurement schemes would correct this problem.

In the metrics arena, we recently conducted a workshop at the Practical Systems and Software Measurement meeting in Orlando, Florida [14]. This enabled us to form a correspondence group who we are working with to develop metrics for at least the following two areas: backlog and test effectiveness. Tracking backlog is important based on our findings because so many projects use it to make their maintenance decisions. Test effectiveness is also important because this is where most maintenance centers spend the majority of their time and effort.

Based on our initial study results, we suggest the following three things that they can do to ease the maintenance burden:

- Put a transition support line item in their Work Breakdown Structure in order to focus attention on and provide a budget for turnover to maintenance.
- Deliver a baselined set of regression tests to your maintenance center for revalidating the baseline once changes are made when you transition the system to maintenance.
- Ensure that licenses for software tools and the software are transitioned with the system to maintenance. Besides potentially cutting tool costs, this practice will enable maintenance personnel to replicate development results in their labs.

This maintenance study will continue until the fall of 2010. Besides the Handbook, we plan to provide recommendations for new metrics and estimating models at that time.

7. Summary and Conclusions

This paper provides a summary of the initial findings, conclusions and recommendations from a year-long software maintenance study being mounted jointly by the U.S. Army and Air Force to improve the manner in which software maintenance is estimated and managed in the field. After some background on the study, ten myths about maintenance were exposed and debunked. Improvements were then explored as the paper looked at ways of levering its findings to make improvements in the way software maintenance is currently being practices in the field today.

The study provides the weapons system community with new insights about how the work that software life cycle support organizations perform is distributed and performed within the weapons systems community. These investigations justify the continued need for further study to determine what the factors are that drive software life cycle support costs and productivity. Once these drivers are known and dealt with, organizational improvements can be made that effectively address software life cycle support during acquisition and development.

The study team is indebted to those who participated during fact-finding. Their openness, support and candid appraisals of current software maintenance practices are acknowledged. So is the support of senior Army and Air Force management within the life cycle software support centers and headquarters.

References

[1] D. Reifer, J. Allen, B. Fersch and B. Hitchings, ***Army/Air Force Software Maintenance Study: What Life Cycle Software Centers Do***, Version 1.6.2, available from Software Engineering Directorate, U.S. Army/AMRDEC, July 2009.

[2] A. April, ***Software Maintenance Management: Evaluation and Continuous Improvement***, Wiley/IEEE Computer Society Press, 2008.

[3] T. Pigoski, ***Practical Software Maintenance: Best Practices for Managing Your Software Investment***, John Wiley & Sons, 1996.

[4] S. Dart, A. Christie and A. Brown, ***A Study in Software Maintenance***, Software Engineering Institute, Report CMU/SEI/93-TR-008, 1993.

[5] K. Erdil, E. Finn, K. Keating, J. Meattle, S. Park and D. Yoon, ***Software Maintenance as Part of the Software Life Cycle***, Tufts University, Dec. 2003.

[6] B. Lientz and E. Swanson, "Problems in Applications Software Maintenance," ***CACM***, Vol. 24, No. 11, 1981.

[7] Institute for Electrical and Electronics Engineers, ***IEEE Standard for Software Verification and Validation***, IEEE Std. 1012-2004, 2004.

[8] J. Allen, B. Fersch, B. Hitchings, J. Judy, D. Reifer and W. Rosa, "Comparative Analysis of Software Maintenance Modeling in the COCOMO II, SEER, SLIM and True S Cost Models," submitted to ***24th International Forum on COCOMO and System/Software Cost Modeling***, MIT, Nov. 2009.

[9] W. Rosa, B. Clark, R. Madachy, D. Reifer and B. Boehm, "Software Cost Metrics Manual," ***Proceedings 42nd DOD Cost Analysis Symposium***, Feb. 2009.

[10] B. Boehm, C. Abts, A. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer and B. Steece, ***Software Cost Estimation with COCOMO II***, Prentice-Hall, 2000.

[11] D. Galorath and M. Edwards, ***Software Sizing, Estimation, and Risk Management***, Auerbach Publications, 2006.

[12] S. McConnell, ***Software Estimation: Demystifying the Black Art***, Microsoft Press, 2006.
[13] R. Stutzke, ***Estimating Software-Intensive Systems***, Addison-Wesley, 2005.
[14] D. Reifer, "Maintenance Measures and Metrics," available at the PSM web site: www.psmsc.com/UsersGroup2009.asp.