Forecasting Software Maintenance

ALEA DESANTIS
KALMAN & COMPANY INC.
Alea.DeSantis@KalmanCoInc.com

ALBERT PLESKUS
SRA INTERNATIONAL
Albert_Pleskus@SRA.com

GARY D. HOWELL
G2 SOFTWARE SYSTEMS
Howell@G2SS.com

MARCH 30TH 2015

1 INTRODUCTION

1.1 PROBLEM STATEMENT

Throughout the Acquisition program lifecycle, a program may find itself in the position of having to defend its budget. There may be frequent drills that demand a response to "What would be the budget impact if your program was cut by X dollars?" During the development and production phases that answer is more readily attainable and explainable: during development, a reasonable answer may be to show that a budget reduction will push a schedule to the right, which may lead to missing an important test date or need by date; in production, it might mean shifting production units and/or fielding schedules farther out. On the other hand, during the maintenance phase of the program lifecycle, and particularly in regards to Software Maintenance, it is often more difficult to assess the impacts of a budget adjustment in a meaningful and quantifiable way. The following method of forecasting Software Maintenance aims to give a program office that tool. What does it mean when you say you will do less Software Maintenance? How does one quantify the amount of Software Maintenance to be performed? What is the impact on the warfighter, end user, and/or mission? Through deep dives performed on actual products, a methodology was brought to light that not only helps answer those difficult questions, but also provides a way to forecast maintenance. It provides a graphical depiction of what predicted Software Maintenance costs are across the lifecycle. It helps answer the question "What does X Sustainment dollars buy me?" and "What level of Sustainment funding best meets user needs *and* is financially viable?" The focus of this paper is how to use actual data on Problem Reports to forecast future Software Maintenance needs.

1.2 PURPOSE

The methodology explained in this paper explores developing a product-specific Software Maintenance estimate based on historical product data. It is a departure from the more familiar methodologies that

are primarily based upon applying a factor, such as defect density rate, to the software size (Software Lines of Code (SLOC)) of a product.

1.3 LAYOUT

This paper is organized into 7 sections:

- Defining Software Maintenance
- Problem Report (PR) Opened Rate
    - Collect Data
    - Analyze Data
    - Forecast Data
- PR Closed Rate
    - Collect Data
    - Analyze Data
    - Forecast Data
- Tools to Present Data and Select Forecast
- Translating Quantity into Cost
- Implementation/Control
- Lessons Learned

2 WHAT IS SOFTWARE MAINTENANCE?

Dr. Barry Boehm, defines Software Maintenance as "the process of modifying existing operational software while leaving its primary functions in tact" (Boehm, Software Engineering Economics, 1981). The Institute of Electrical and Electronics Engineers (IEEE) Standard for Software Maintenance, IEEE 1219, -- defines it "as the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment." (Institute of Electrical and Electronics Engineers, 1998). The Defense Acquisition University (DAU) describes the Software Maintenance Cycle as the process of sustaining the "software product throughout its operational lifecycle. Modification requests are logged and tracked, the impact of proposed changes is determined, code and other software artifacts are modified, testing is conducted, and a new version of the software product is released." (DAU, 2005).

2.1 PROBLEM REPORTS (PR)

Logged and tracked requests can be referred to as Problem Reports (PR); however a PR can have many names: Trouble Ticket, Software Anomaly Report, Software Trouble Report, Defect, Software Deficiency Report, Bug, etc. For the remainder of this paper, Problem Reports (PR) will be used. PRs enable the quantification of Software Maintenance. A PR is typically submitted by a consumer/user to describe a perceived deficiency in a Software product, and is usually initiated upon the discovery of a functional or operational defect.

Problem Reports can be discovered through multiple avenues. A primary avenue for discovering issues and generating PRs is through the test process. This test process can include: lab, regression, Design Verification Testing (DVT), System Operational Verification Testing (SOVT), Development Test

(DT/OT/FOT&E), System of Systems (SOSI), End to End (E2E), etc. A second avenue of PR detection is during (and after) system fielding, where issues are discovered by either users or those providing software technical assistance and/or support.

For the purposes of this study, a formal Problem Report would only be recorded after the product has completed a formal verification test and has been approved for release by the program's Configuration Control Board (CCB). There will be PRs that are discovered along the Software development process, but that's not used in this analysis, as it ties to the development phase. This study defines the beginning of the maintenance period as when the product has formally closed out development (Sell-Off testing) and has entered the field or operational use. A backlog may remain from the development phase, and it's important to capture that backlog (use as a starting point for forecast), but for all trend analysis purposes, that data should not be used.

3 PR Opened Rate

3.1 Step 1: Collect Data

The first step is to collect the data. Data collected should observe the number of PRs that have been opened against a Software product. Data analysis should be done at either the total Product level, or if possible/practical (data allows), by each major software version released (major release relates to an actual increase in functionality, not a software patch that only fixes bugs).

3.2 Step 2: Analyze Data

A) Plot the number of PRs opened each month (# PRs Opened over Time (Months)).

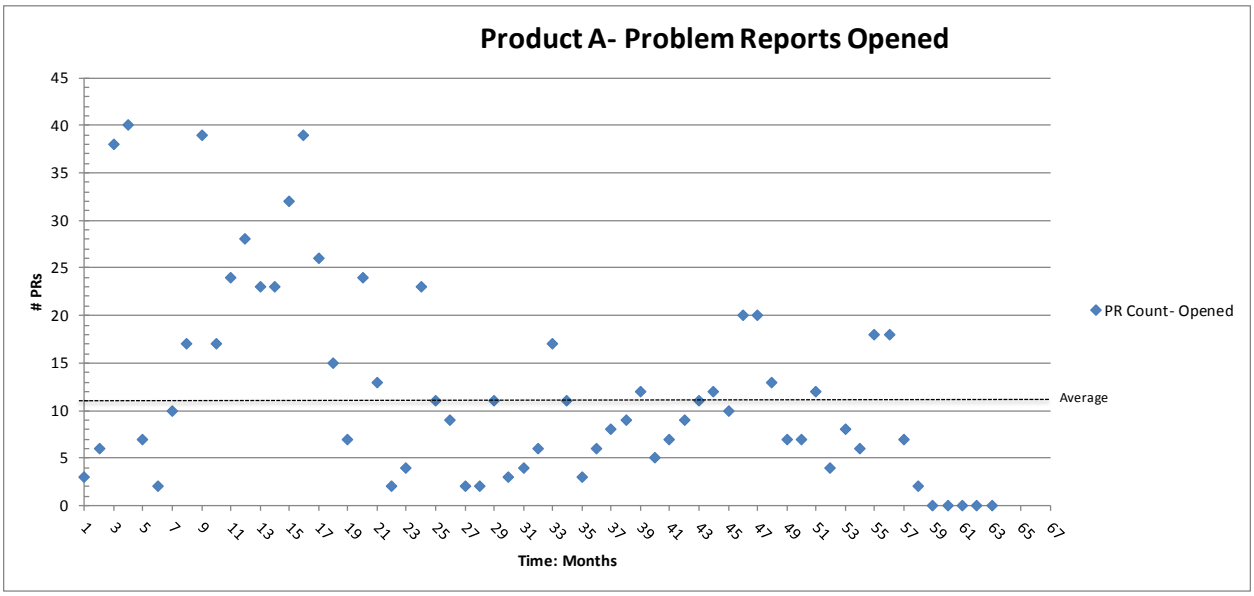


Figure 1: PR Opened Scatter Plot

B) Overlay product lifecycle events (Test Events, Fielding events, version releases, etc.). By overlaying the product lifecycle events, you can see if there is a correlation in spikes of PR counts with respect to specific events. Remember, for trend analysis, use only the data that is

after the Sell-Off Acceptance Test. Data from different products, has shown that spikes of varying magnitude occur around the following types of events: Initial Release of product, Version Releases, Fielding events, and Test Events.

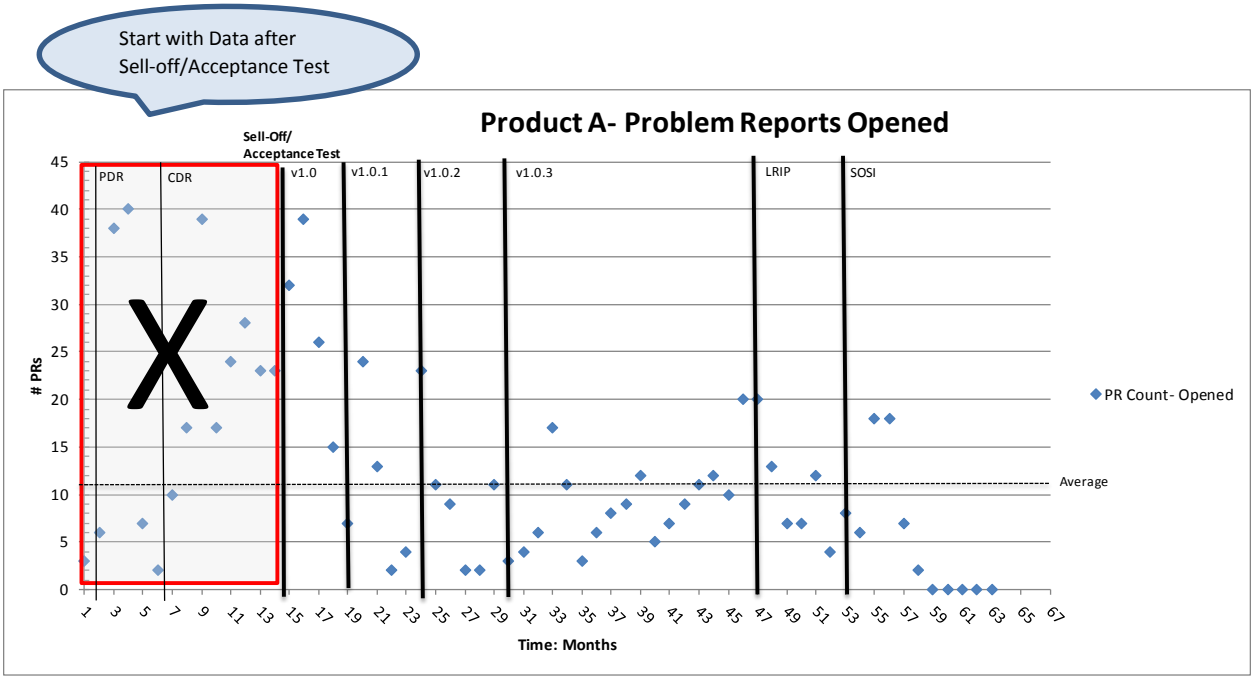


Figure 2: PR Opened & Lifecycle Events

C) Through statistical analysis (detailed in Step E), you can stratify the data between Special Cause Variations and Common Cause Variations. Common Cause Variation is defined as a constant system that captures the expected amount of natural variation in the process. *(Thomas Pyzdek, 2010)*. Special Cause Variations are "caused by a source of variation that is not part of the constant system…. The basic rule of statistical process control is variation from common-cause systems should be left to chance, whereas special causes of variation should be identified". *(Thomas Pyzdek, 2010)*. The Special Cause Variations are identified by these event driven spikes. By removing the Special Cause data points, the resulting stratified data set allows you to ascertain the typical average monthly PR Opened rate. This stratification is Common Cause Variation.

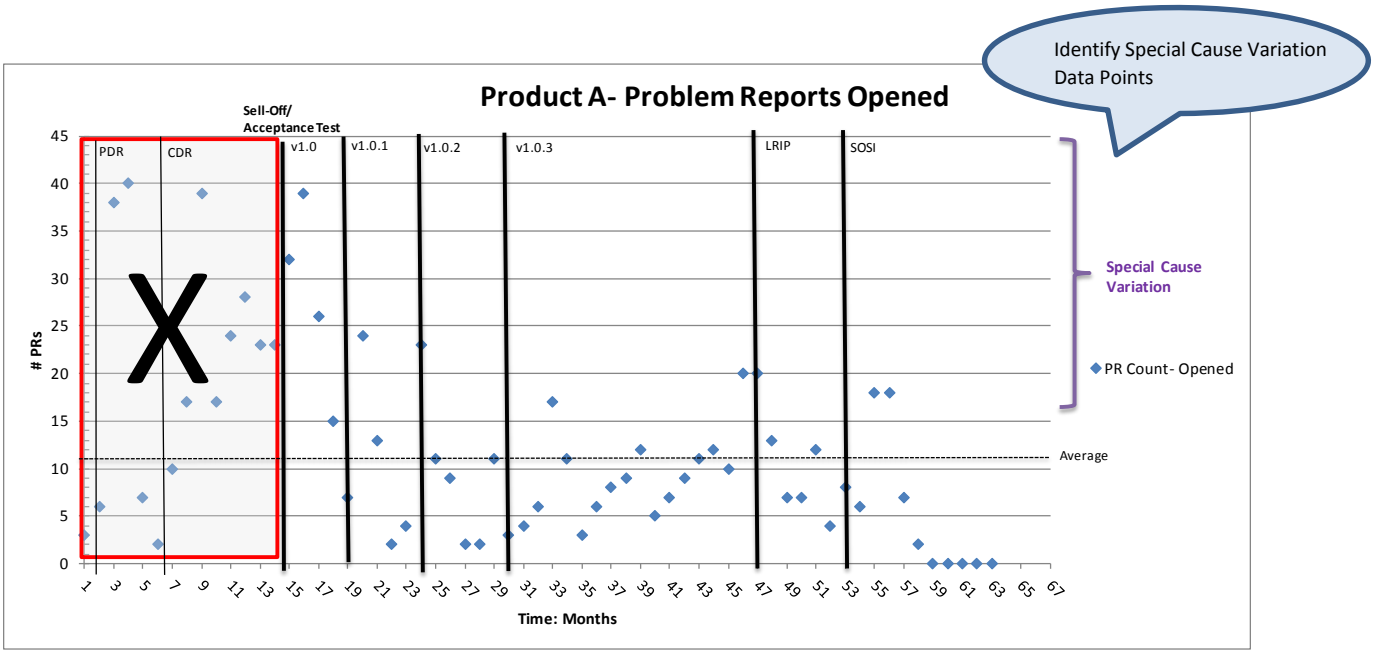


Figure 3: Special Cause Variation

D) Common Cause Variation accounts for typical variance in monthly counts.

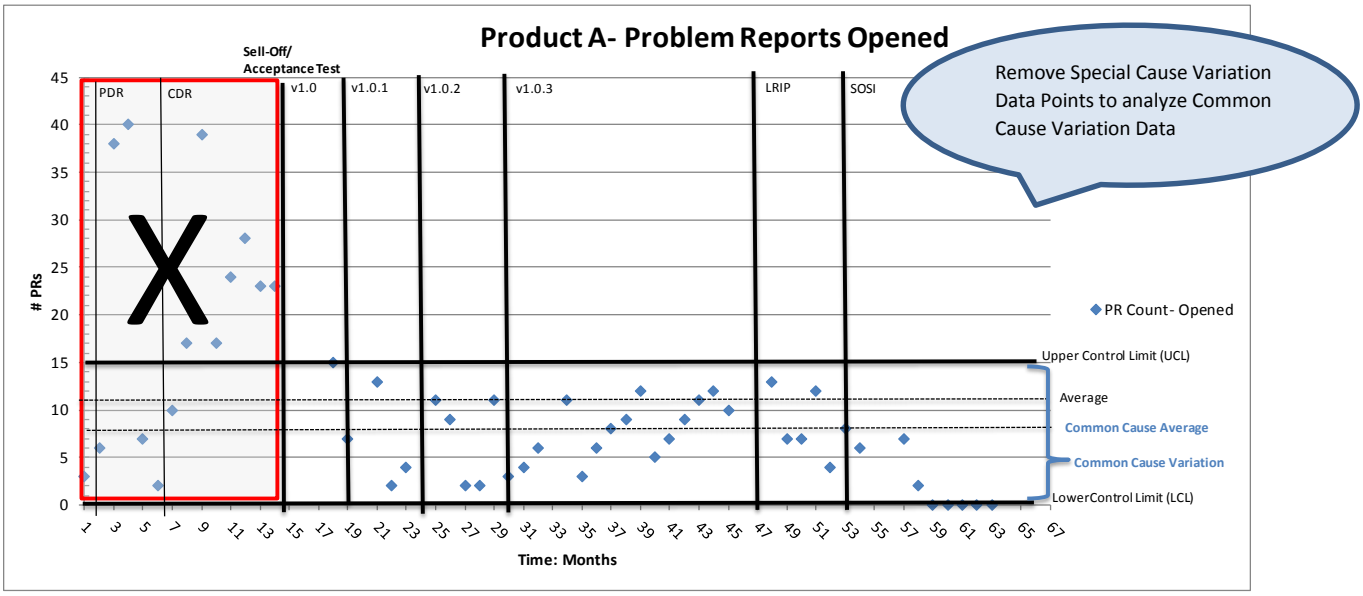


Figure 4: Common Cause Variation

E) Several different statistical process control charts (frequency charts, run charts, and control charts) (Thomas Pyzdek, 2010) are used to determine control limits for the Common Cause Variation and to ensure the data is correctly bounded. The Run Charts and Control charts use only the Common Cause data points.

Run Charts are used to identify problems (special causes) and when those special causes seem to be occurring. The following run chart shows the "random variation associated with the process driven by common cause variation". (P.Pande, 2002).

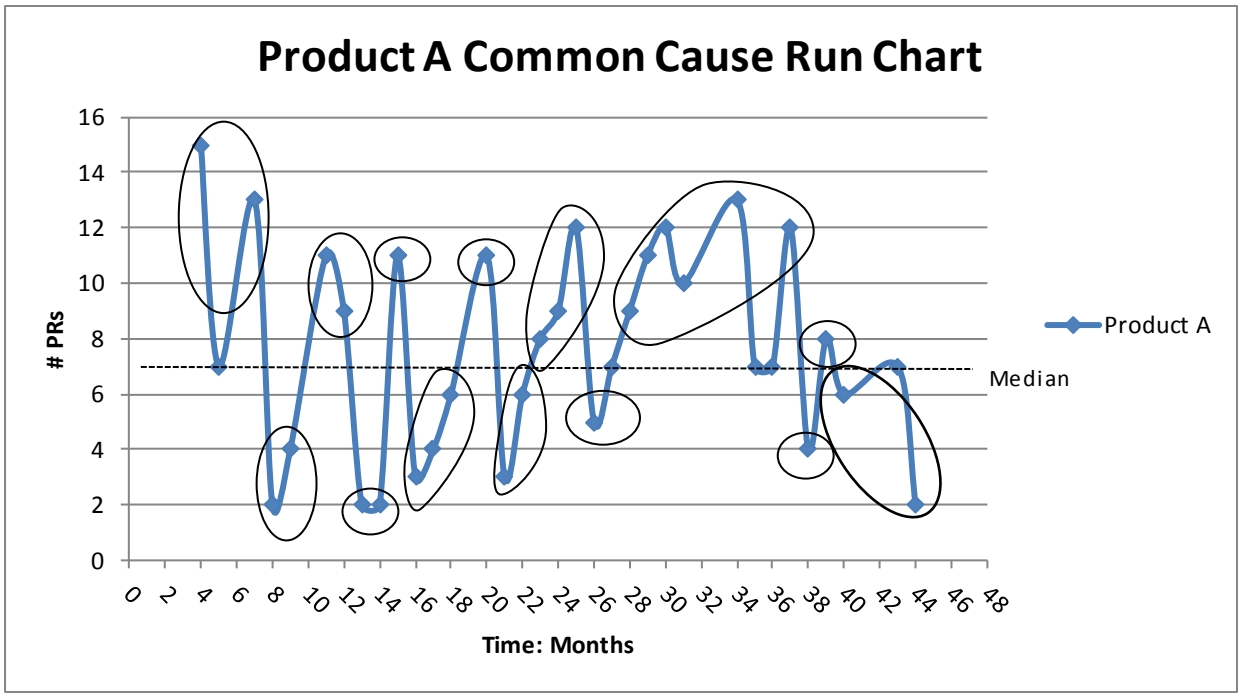


Figure 5: PR Opened Common Cause Control Chart

Control Charts are advanced Run Charts. They can show the expected amount of natural variation in the process. (P.Pande, 2002). It can identify the data points that fall outside of the control limits, and therefore cannot be considered to be a result of the natural variation within a process. The central limit theorem states that "stable (no special causes) distributions produce normally distributed averages, even when the individual data are not normally distributed". (Thomas Pyzdek, 2010). Use 3 Month Average data and set control limits at (+/-) 3 standard deviations. (Thomas Pyzdek, 2010).

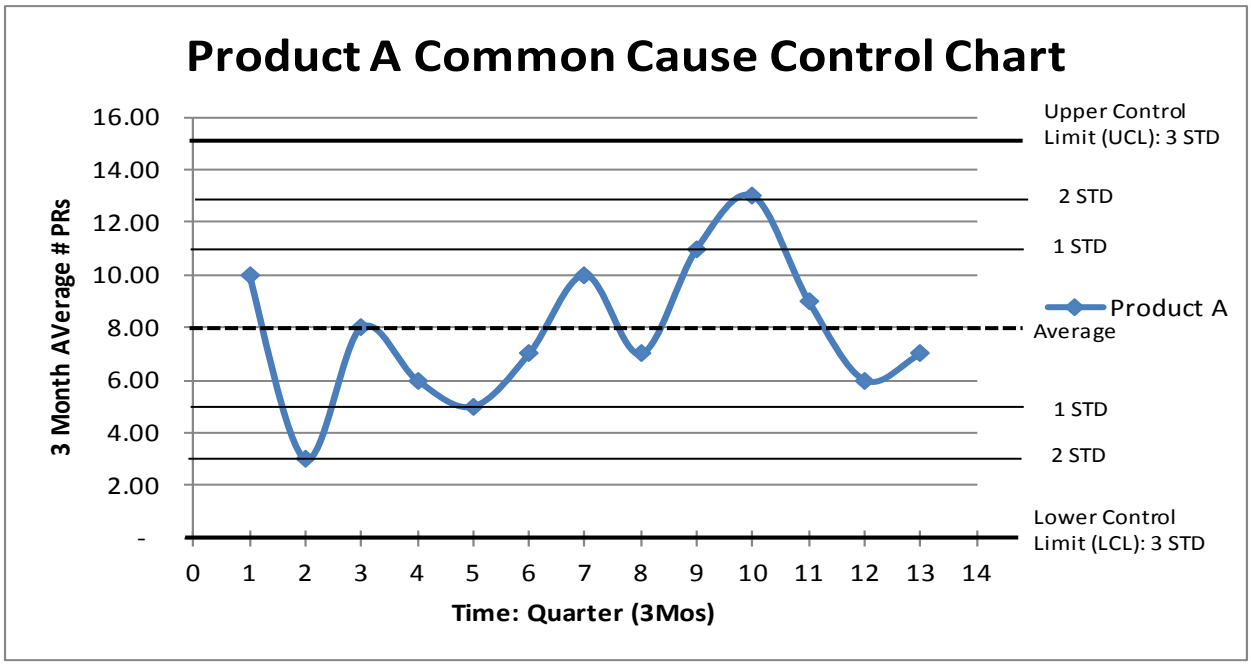


Figure 6: PR Opened Common Cause Control Chart

A frequency chart presented as a histogram, provides a better view of the center, distribution, and shape of the data, as well as how much variation there is. (P.Pande, 2002). Label Common Cause Variation bounds (based on Run and Control Charts). Label Special Cause Variation events by referring back to the events identified in the scatterplot in Section 3.2.B.

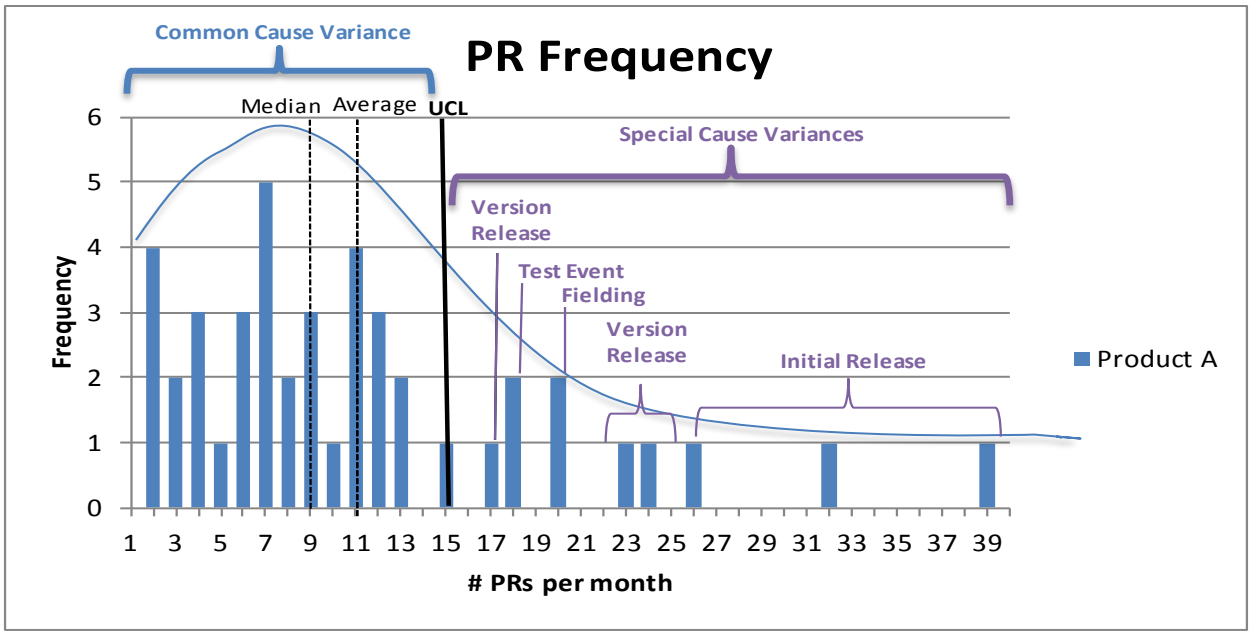


Figure 7: PR Opened Frequency Chart

Cumulative Frequency Charts represent the sum of all frequencies up to and including that value. (Thomas Pyzdek, 2010). This graph helps you understand how using solely a monthly PR rate, will only capture X percent of the data. It emphasizes the need to additionally capture the Special Cause Variations in the forecast model, allowing for a more complete analysis. For example, if the forecast for Product A is only modeled after a monthly PR Opened rate of 8, it will only capture 45% of the data.

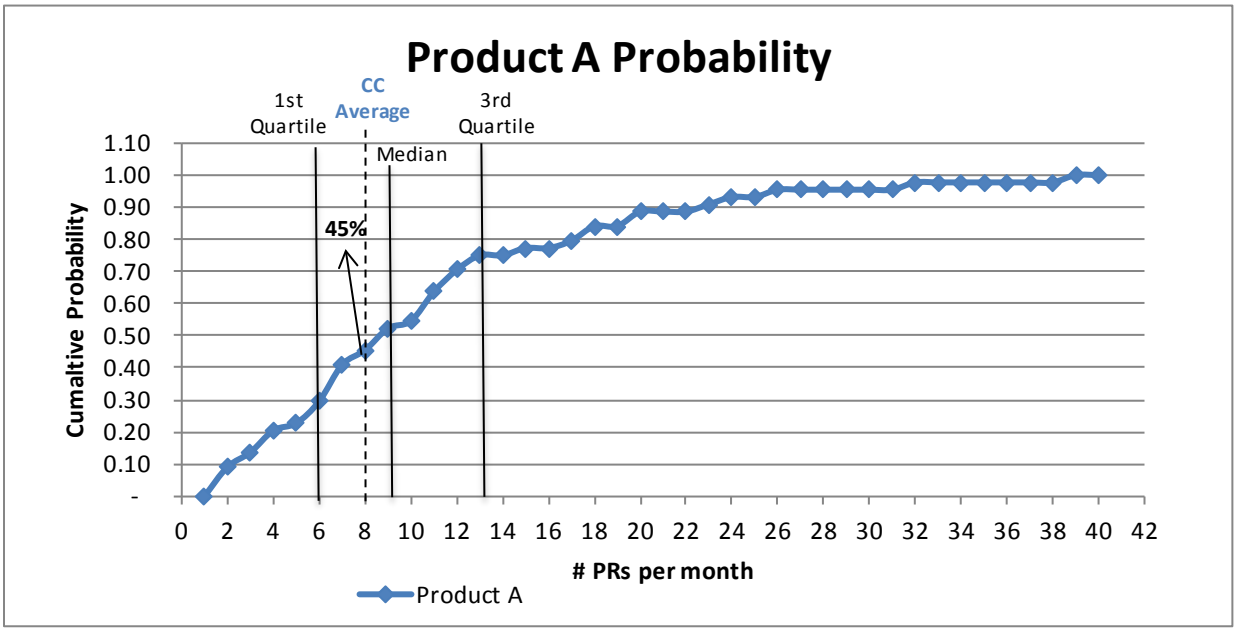


Figure 8: PR Opened Cumulative Frequency Chart

F) Create final scatterplot of PRs Opened over Time, and highlight trends discovered.

- Plot PRs opened over time
- Overlay program's lifecycle events
- Overlay Common Cause Bounds and Average
- Label Special Cause Variations

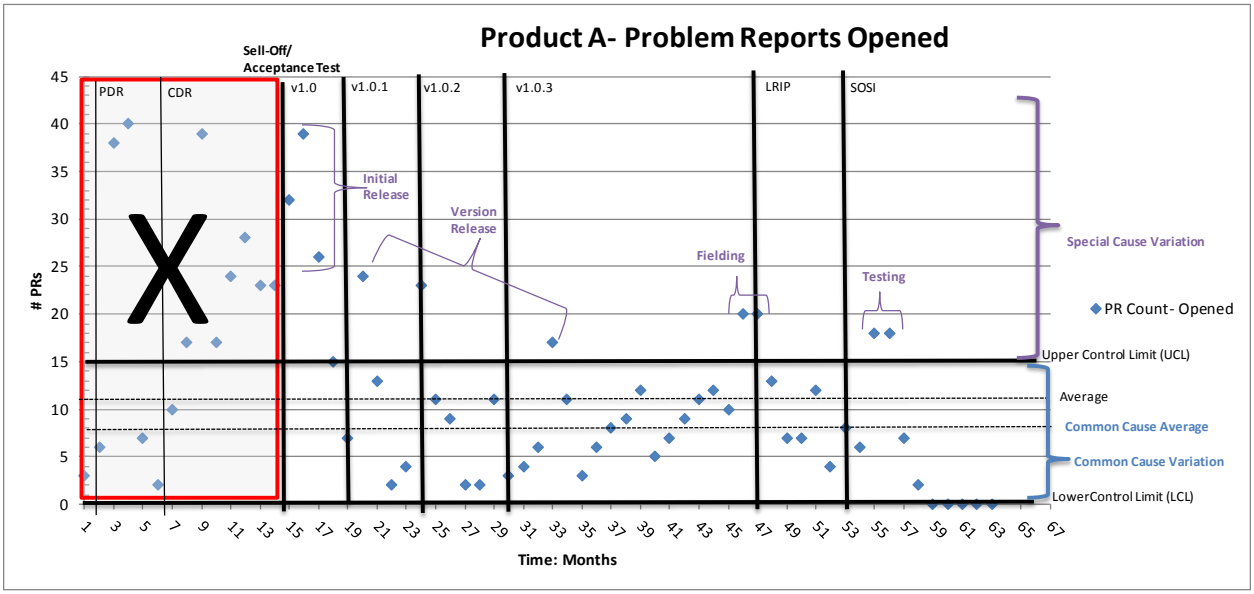


Figure 9: PR Opened Trend Analysis

3.3 STEP 3: FORECAST DATA

A) Model Common Cause Variation. To Forecast PRs over time, you need a monthly PR Opened rate. This was determined during the analyze phase and is the average from the Common Cause Variation data. Monthly PR Opened Rate = Common Cause Average

B) Align known future event dates (Test Events, Fielding events, version releases, etc.). Around these future events there should be a spike in PRs Opened as seen from past experiences. Factors can be developed from what was recorded around the prior events. For example, say around a test event, the spike in PRs increased from the Common Cause Average by a factor of 3 for 2 months; around fielding the spike increased by a factor of 2 for 3 months; around a version release, the spike increased by a factor of 3 for 1 month; etc.

C) Apply these factors to the modeled Common Cause Average at the point in time where these known events are expected to happen. The resulting forecast becomes the baseline PRs Opened.

D) Determine a Point of Decay for the rate of PRs being opened. As a product matures over time, and more PRs are identified and fixed, the PR Opened rate should taper off. Determining this Point of Decay can be challenging, especially when historical data is not present. Each product has its own Point of Decay based on assumptions specific to that product. Identify what the events are that determine maturity: I.e. Full Operational Capability (all users have had a chance to exploit the product; System Operational Verification Tests (SOVT)-derived PRs will fade); or the expectation that the full capability of the Software will have been utilized; or all known enhancements will be realized. Once the Point of Decay is established, the PR Opened rate will

decline at a rate of X% per year until it settles at the average minimum monthly rate of *n* PRs Opened per month (minimum rate determined by historical data). By looking at current data, you may be able to calculate a factor for what that decay rate may be. For example, Y% of all PRs were opened as result of SOVT. Therefore when FOC is reached, the amount of PRs Opened should drop by Y% and then continue to decline each year.

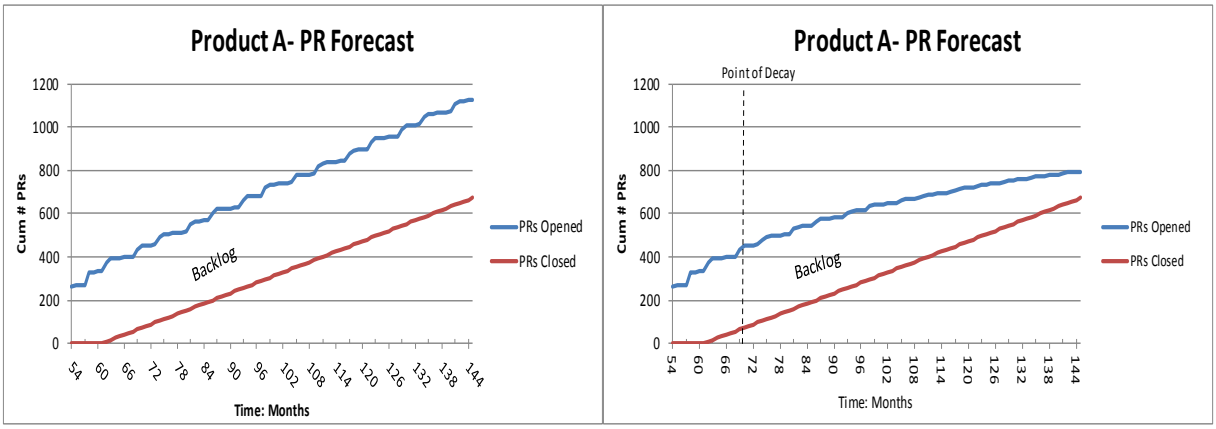
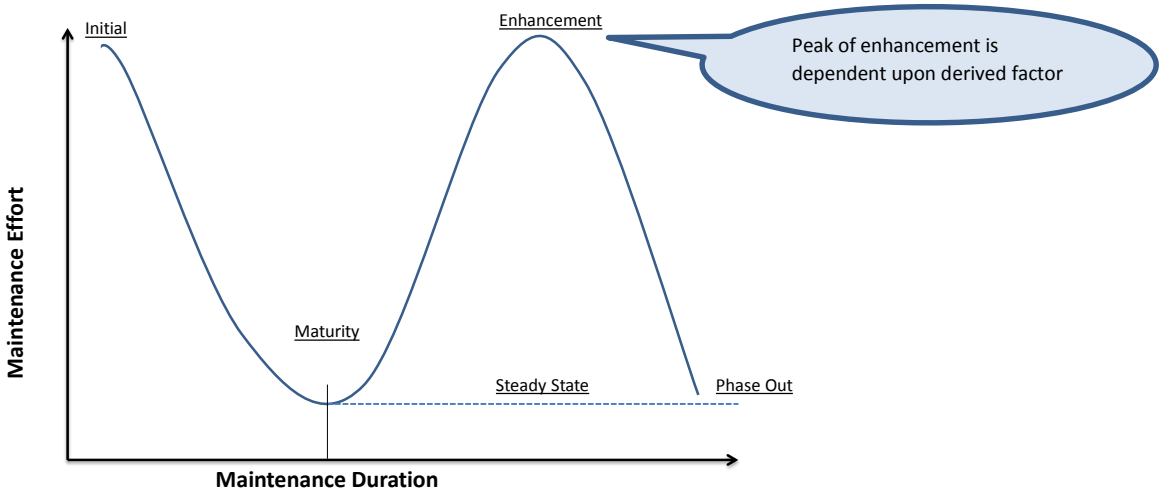


Figure 10: Left: Without Decay; Right: With Decay

E) The exception to a decay curve is if/when a significant capability is added to the Software product, in which the entire PR Opened cycle repeats itself but at a proportionally lesser scale. Factors can be developed to proportionally scale this enhancement (e.g. Product A (baseline) SLOC was 100K SLOC, Product A (enhancement) is estimated to be 10K SLOC, proportional scale is 10%). In absence of data, factors can be derived from close talks with subject matter experts.



F) By applying a decay curve to the baseline PR Opened curve, and adding in known future enhancements, you now have your complete PR Opened Curve.

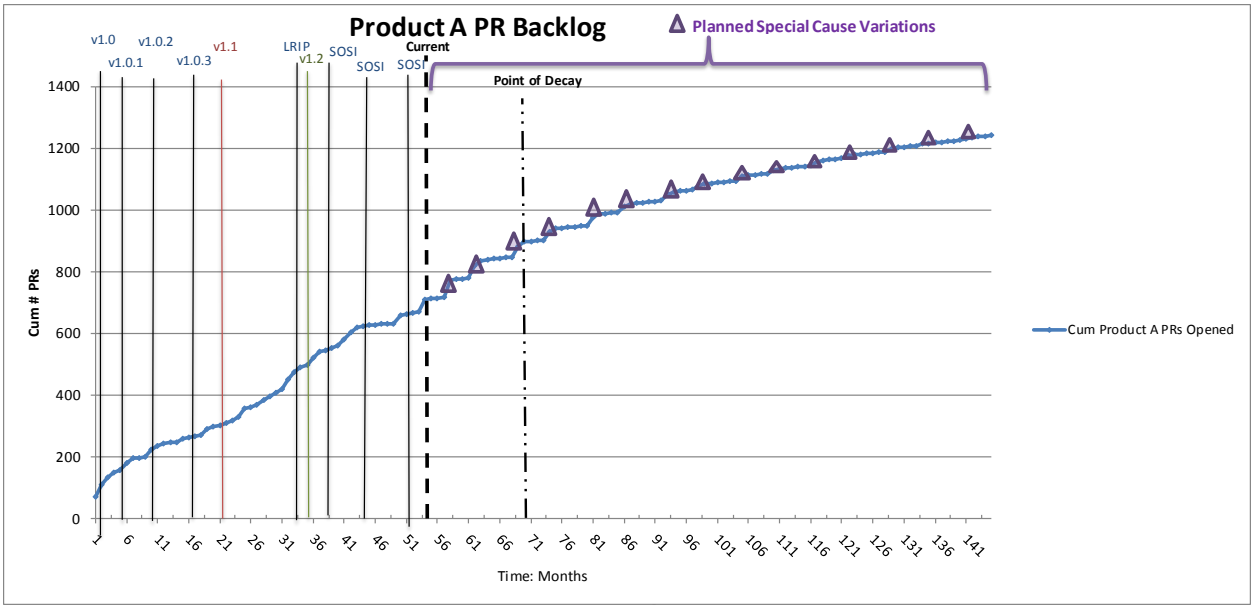


Figure 11: PR Opened Forecast

4 PR Closed Rate

4.1 Step 1: Collect Data

Data collected should observe the number of PRs that have been closed against the Software product.

4.2 Step 2: Analyze Data

The same steps should be taken when analyzing PRs Closed as it was for PRs Opened. Separate out Common Cause Variation from Special Cause Variation. With the PR Closed rate, you want to solely focus on the Common Cause Variation. The peaks in the closure rate are only important for removing them to get to the underlying Common Cause Variation. The Special Cause Variations will not be used in the forecast (discussed below). With the remaining data, determine what the monthly average closure rate is. Depending on how data is collected, you may see closed PRs lumped in big groups around a version release date, or you may see them being closed each month. In the case of PRs being closed in large chunks at time, (for example 0 closures for 6 months plus, and then a large number all at once), look at either using a 3 month average or (longer duration) to get a true idea of what the rate per month would be.

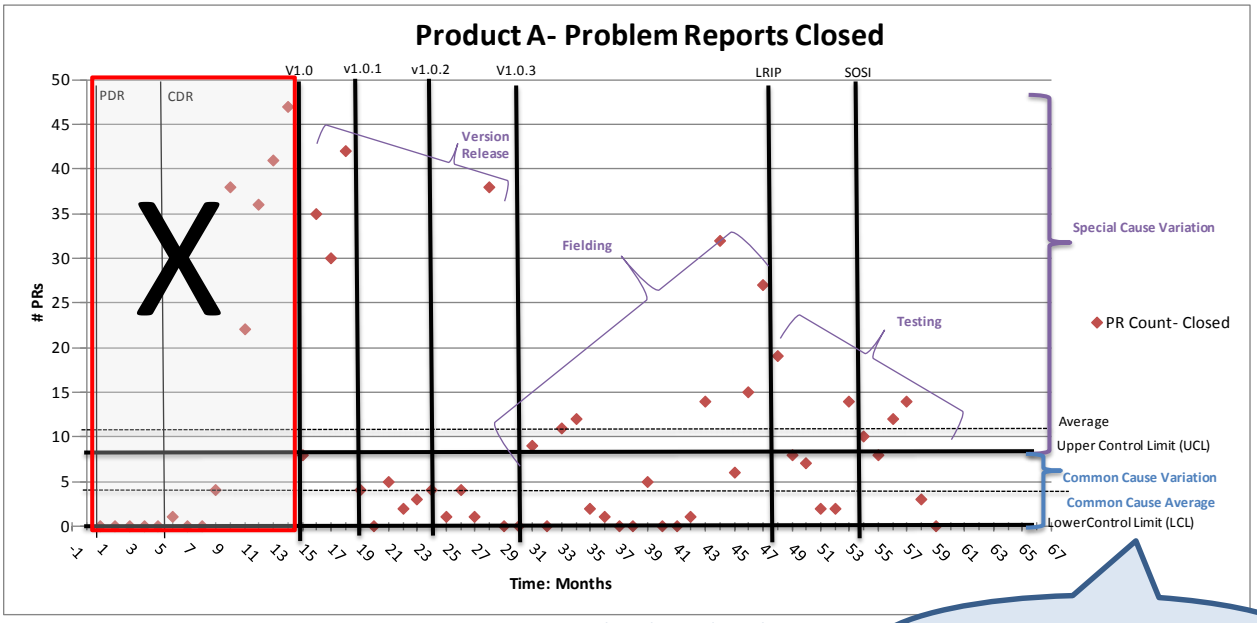


Figure 12: PR Closed Trend Analysis

4.3 Step 3: Forecast Data

The PR Closure rate is a causality dilemma, of which came first: the budget or the number of PR fixes; you can only close as many PRs as you have funds to do. Consequently, when developing a prediction model, there are endless scenarios moving forward. What matters is the balance between affordability and the amount of acceptable PR backlog (Backlog Management Index (BMI); discussed in following section). Sample Scenarios to look at could be: Scenario 1) Current Budget: Amount of PRs that can be closed within the current Maintenance Budget (fiscally constrained); Scenario 2) Historical Rate: Amount of PRs Closed based on past performance (still partially fiscally constrained); Scenario 3) Closing PRs only with a Severity level of 1-3 (requirements perspective; severity levels addressed in following section); Scenario 4) Achieving and maintaining a desired BMI level (done at either the total level, or for specific severity levels) (fiscally aware & requirements perspective). Presenting several scenarios, gives leadership a way to make trade-offs to come up with a solution that balances both funding and requirements (is financially viable yet meets user needs). Tools to help present this data are described in the following section.

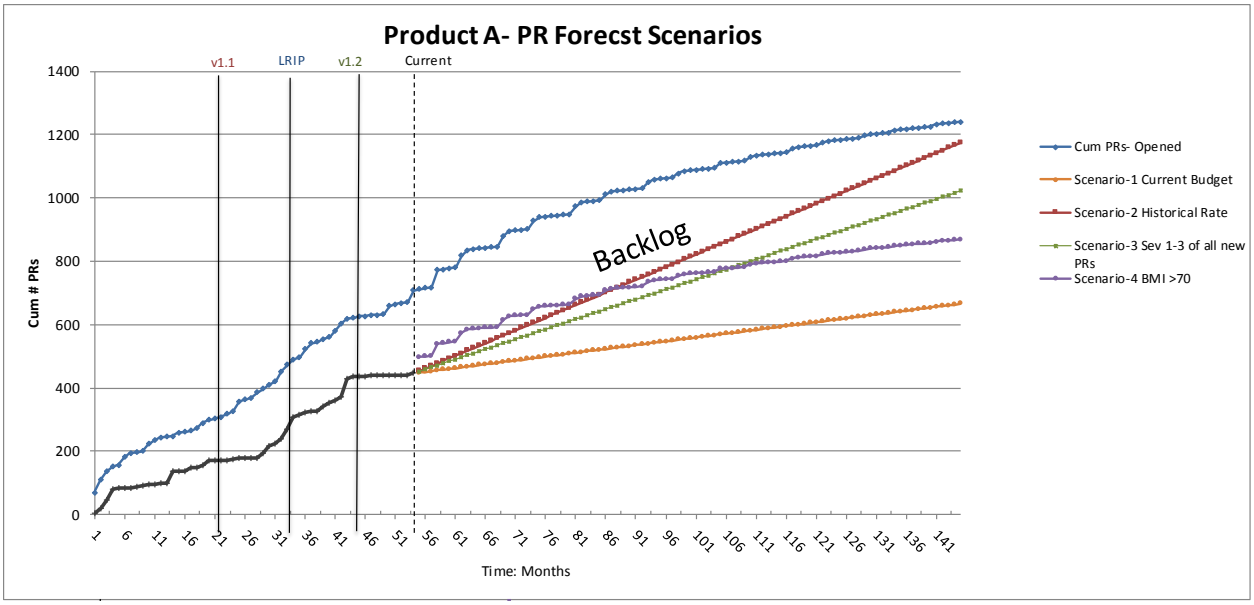


Figure 13: PR Forecast Closed Scenarios

Due to the nature of modeling the closure rates, the closed curve will be smoother and possibly linear. In the real world, PRs will close more like what's been seen historically (ups and downs). Therefore, if comparing forecasted PR Closure rate against tracked actuals, look at longer periods of time, versus an individual month.

5 TOOLS TO PRESENT DATA AND SELECT FORECAST

5.1 PR BACKLOG

Obtaining metrics on both PRs Opened and PRs Closed allows the calculation of PRs Backlogged. PRs Backlogged represents the count of PRs that will remain unfixed during a given time frame.

PRs Backlogged= # PRs Opened – # PRs Closed

This depicts exactly how much maintenance will be done and how much is left on the table.

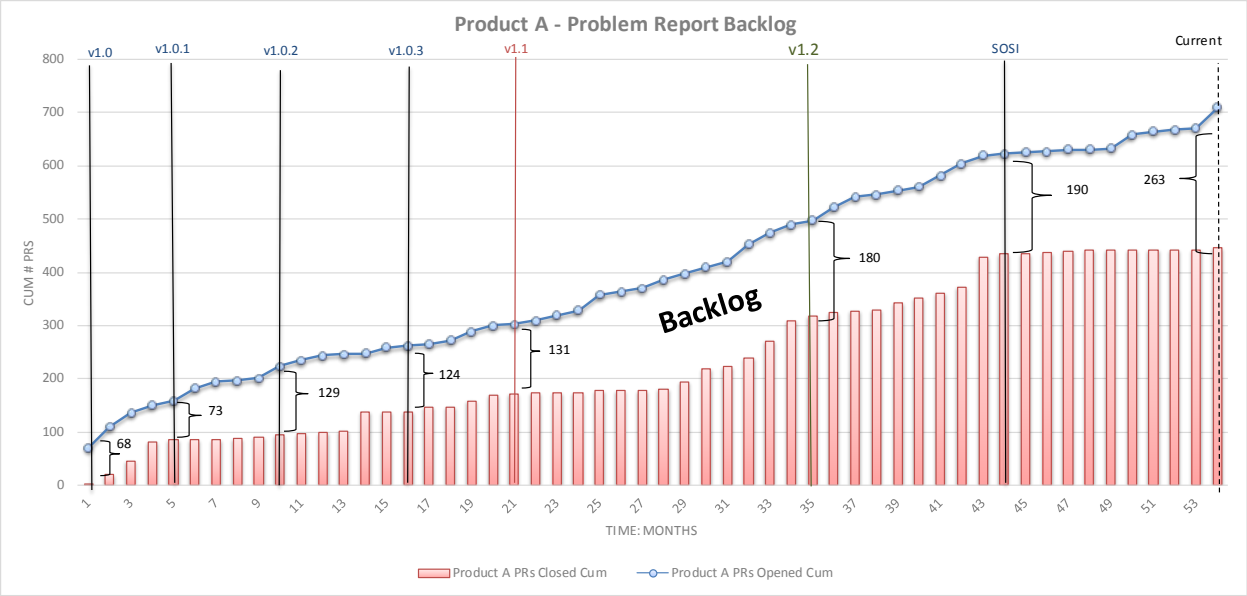


Figure 14: Problem Report Backlog

5.2 Backlog Management Index

The Backlog Management Index (Kan., 2002) is a metric designed to signify the amount of fix responsiveness (how am I managing my backlog).

$$\text{BMI} = \frac{\text{Number of problems closed during the month}}{\text{Number of problem arrivals during the month}} \times 100\%$$

Figure 15 Backlog Management Index

If the BMI is greater than a 100, it means the backlog is being reduced in that month. This is another great metric that can visually depict and quantify what exactly Software Maintenance means, and what the impact is if not funded to a certain level. It can also be used to map out a specific closure scenario. For example, "PR Closure rate needs to be at a level to maintain a BMI of 70 or greater".

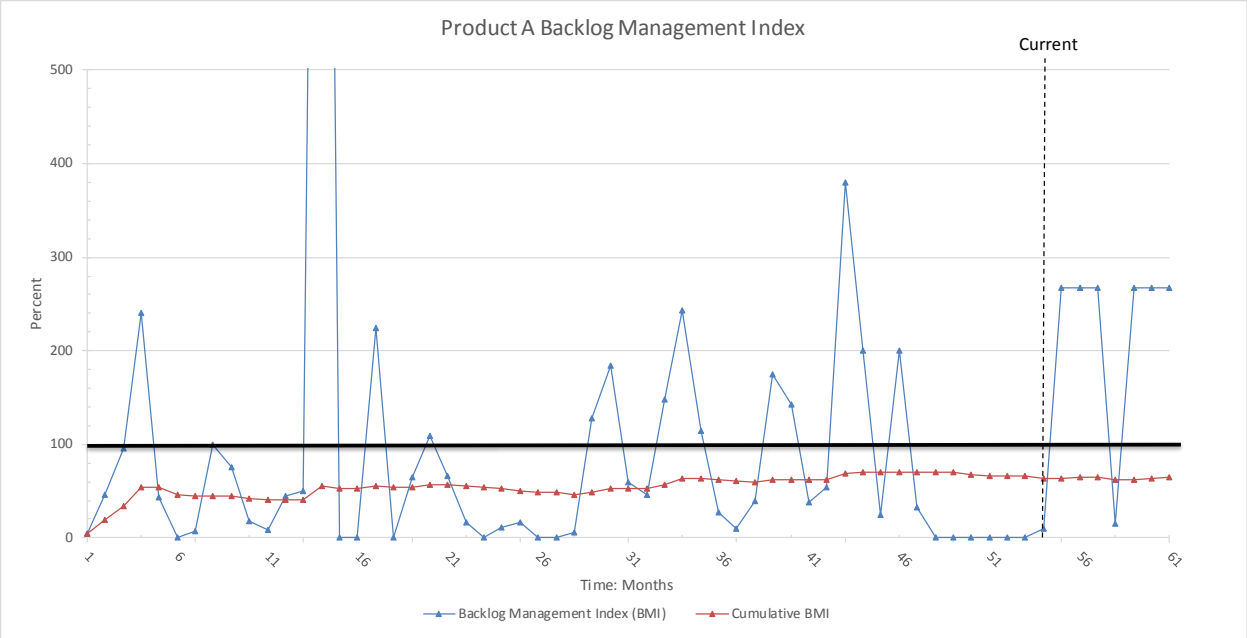


Figure 16: Backlog Management Index

5.3 Average Duration to close a PR

Data collected can be used to display what is the average duration a PR remains opened before it is closed. This metric could be used as a benchmark or a way to model a scenario. For example, "All PRs need to be closed out within 180 days of being opened".

5.4 Severity Level

The reality is, not all PRs will be closed, and not all will need to be closed. Some will not be important enough, and others will not be affordable. A way to prioritize needs is by segregating the data by severity level.

PRs are typically categorized by severity level (IEEE, 2009); ranging from 1 (Critical) to 5 (least severe). The following definitions are common:

- Severity 1- "Critical: Prevents Essential Capability"- Issue likely to result in a failure that affects/jeopardizes safety of operator, personnel, or the system itself. The issue prevents the accomplishment of an operational or mission essential capability. This severity level also includes most security-related issues.
- Severity 2- "Major: No workaround"- Issue results in failure of one or more channels within the system (application failure). Affects the accomplishment of an operational or mission-essential capability and no work-around is known.
- Severity 3- "Adverse: Workaround"- Issue does not result in a system failure, but causes system to produce incorrect, incomplete, or inconsistent results; adversely affects the accomplishment of an operational or mission-essential function. System usability is impaired, but a documented work-around is available.
- Severity 4- "Minor: Operational Inconvenience"- Issue does not cause a failure or impair usability; results in user/operator or development/support personnel inconvenience or

annoyance but does not affect required operational or mission essential capabilities or accomplishment of development/support responsibilities.

- Severity 5- “Exception: Other Effort”- Non-conformance to standards, generally aesthetic issues. Also includes changes to documentation.

Adjusting the charts described above by severity level provides the ability to run scenarios based on priority. PRs can be segregated into what absolutely needs to be fixed and what could be delayed. This can highlight potential significant concerns in the future, in the case that a program office doesn’t have enough funding to fix the critical PRs. Tracking by severity level brings to the forefront, the need to focus on the critical PRs, and not just the easy-to-fix bugs. It highlights not only what type of problems are being found, but what type are being closed.

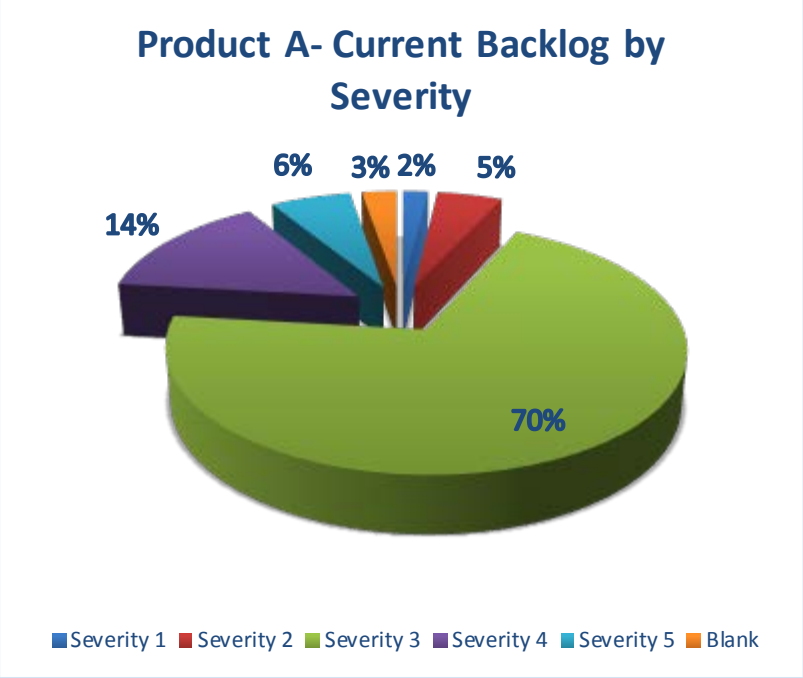


Figure 17: Backlog by Severity

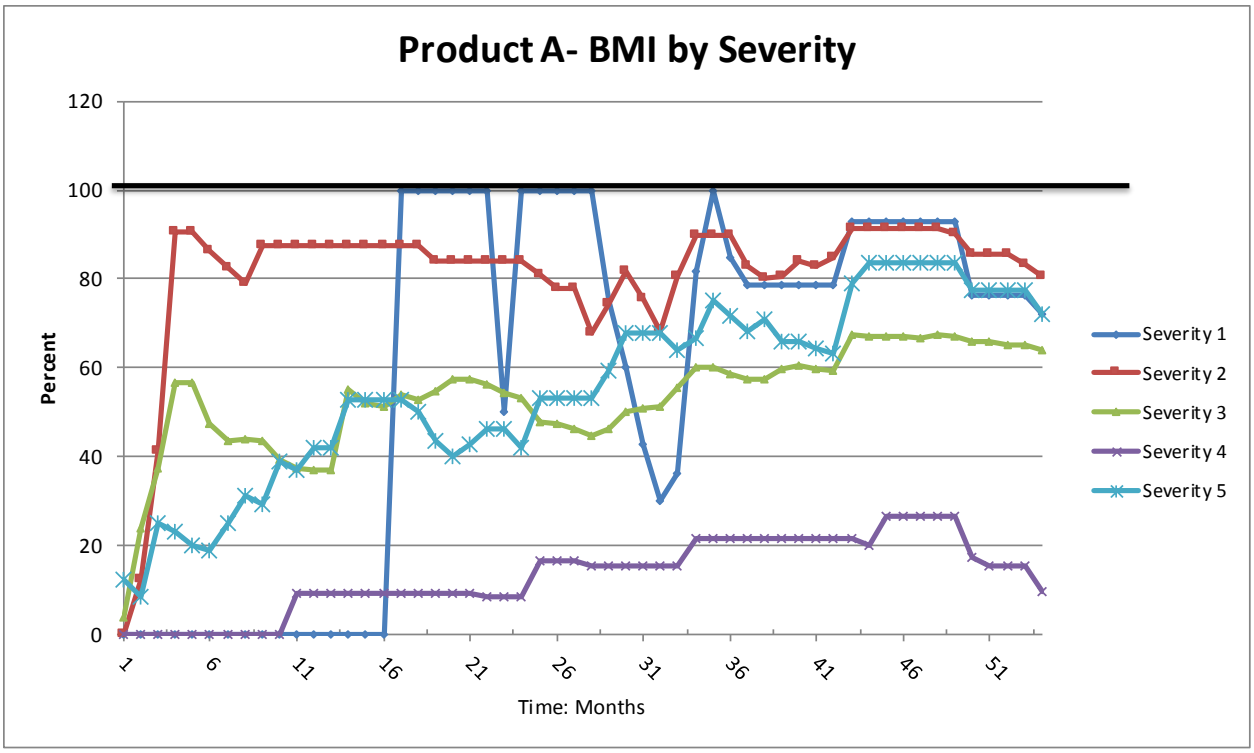


Figure 18: BMI by Severity

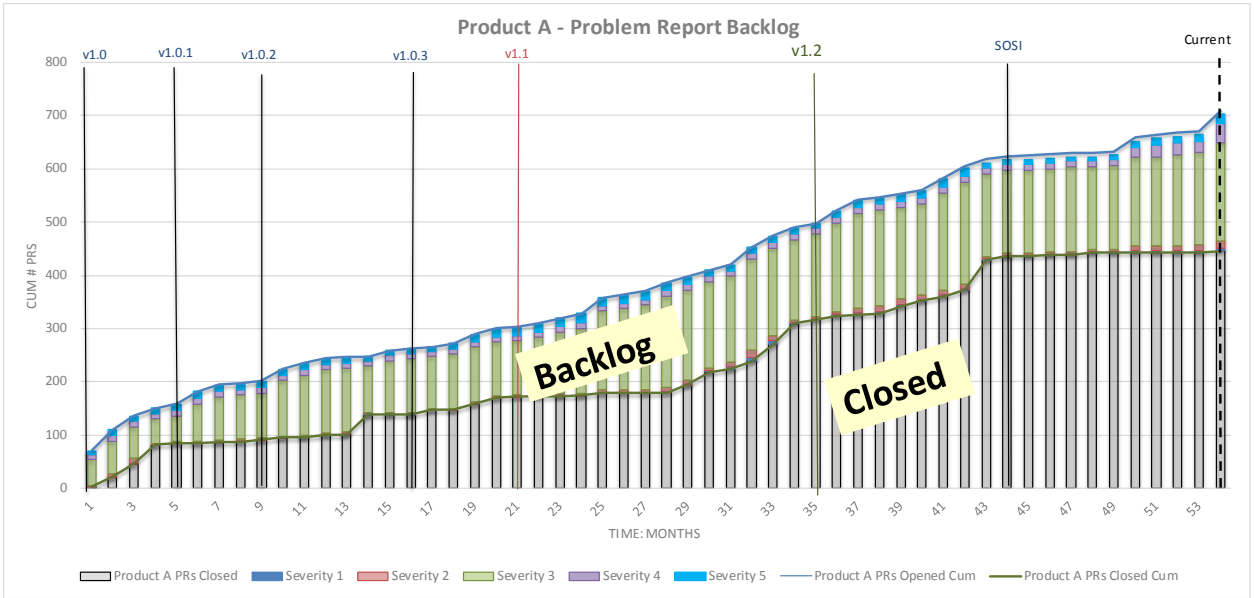


Figure 19: Time Phased Backlog by Severity

6 TRANSLATING QUANTITY INTO COST

Once the quantity of PRs is forecasted, costs need to be calculated, either by:

1) Applying Hours per PR fix, and then applying labor rate
2) Or applying Cost per PR fix

The method chosen depends on data available for analysis.

6.1 METHOD 1: HOURS PER FIX

Deep dives conducted on different Software products indicated hours fell under three main categories: Direct Hours to fix a PR; Testing, Integration, & Software Support hours; and System Engineering and Program Management Hours (SEPM). The results are summarized below. When applying a cost per PR, be mindful as to what that cost includes to ensure there is no double counting. In other words, if Integration/regression testing is accounted for in another bucket, don't account for it in here.

6.1.1 Direct PR hours

6.1.1.1 #Direct PR hours per PR

6.1.1.1.1 Direct PR hours per PR by Complexity

Results from the deep dive concluded that Direct PR hours per PR can vary by the Complexity Factor of each PR. The Complexity Factor is defined as the size of effort (# hours) required to fix a PR. Some PRs could take only ~30 hours to complete, while others could take hundreds of hours to complete. If enough data exists, you can break up the extensive range of hours, and develop a scale that provides a range of PR hours by Complexity Factor. After analyzing the data among a handful of products, the Hours per Complexity Factor were determined using the following equation: Complexity Factor 1 is 30 hours. The starting point of hours for Complexity Factor 2, would be 1.5 times the amount of hours in Complexity 1. The starting point of hours for Complexity Factor 3, would be 1.5 times the amount of hours in Complexity 2, etc. It's important to identify when a PR fix becomes more than just a fix, and should really be classified as a perfective enhancement. The deep dive indicated that the most a PR fix would encapsulate is ~1000 hours. Anything beyond that would need to be categorized as perfective enhancement. Therefore, for Complexity Factor 10, the starting point was lowered to 900 hours and ranged to 1000.

Average PR Hours	
Complexity	**Proposed Hrs**
1	30
2	45
3	68
4	102
5	153
6	230
7	345
8	518
9	777
10	900

Figure 20 Complexity Scale

Once a scale is determined, it can be validated in two ways. First, look at each individual PR to determine how many fell within the bounds proposed for each complexity factor. Determine if that % is an acceptable % of data falling within the range (confidence level). Second, look at how accurate using this scale would be in estimating hours. Use the proposed scale and apply that as the calculated hours for each PR. Then compare the resulting Total PR hours to the actual PR hours. This assesses how close those calculations are to the actuals. Next determine if that is an acceptable %. In our example, the resultant measure of accuracy for using a proposed hour scale for depicting Direct Hours per PR was 1.05.

	Average	Product 1	Product 2	Product 3	Product 4	Product 5
Factor to apply to PR table to match actuals using complexity scale:	**1.05**	1.31	0.88	0.89	1.19	0.96

Figure 21 Accuracy of Complexity Scale

6.1.1.1.2 Average Direct PR hours per PR

Determining the Complexity Factor for future unknown PRs, is a challenging task. Calculating "Direct PR hours per PR" across all complexity levels, provides an average factor. To validate the accuracy of using this factor, check to see if the hours align with the average complexity in the scale developed earlier. For example if the average Complexity Factor is a 5, ensure the Average Direct PR hours align with the proposed range of direct hours associated with a complexity level of 5.

	Average Hours per PR					
Factor	**Average**	Product 1	Product 2	Product 3	Product 4	Product 5
Average Hours per PR	**183**	170	210	153	234	150
Average Complexity	**5.00**	5	5	5	5	5

Figure 22 Average Direct Hours per PR

Next determine how accurate using an average PR rate would be in estimating hours. Apply the Average Direct PR Hours per PR. Then compare the resulting Total PR hours to the actual PR hours. This provides a measure of accuracy on the Average Direct Hours per PR.

	Average	Product 1	Product 2	Product 3	Product 4	Product 5
Factor to apply to PR table to match actuals using an Average Hours per PR:	**1.04**	1.08	0.88	1.21	0.79	1.23

Figure 23 Accuracy of Average Direct Hours per PR

Determining the Complexity Factor for the future unknown PRs is challenging. If the results between using a PR hours by complexity versus using an average of PR hours were close, using the Average Direct Hours per PR is the preferred choice. However the risk of moving forward with this assumption is that future PRs may not be on average a complexity of 5. Risk should be applied to handle this uncertainty.

6.1.2 Other Hours

We found that there were two types of "Other Hours" that were contained within the Total Hours per Task Order. There were Testing, Integration, and Software Support hours, as well as typical SEPM hours. Both of these sets of hours need to be taken into account when calculating total hours per PR.

"Cost Estimating Relationships (CER's) are mathematical expressions relating cost as the dependent variable to one or more independent cost driving variables." (Stuparu, Tomita, & Stanciu, 2010). Ensure

analysis is done to develop CERs using Direct PR hours as the basis (the independent variable). The end result is to apply a factor on top of the Direct PR hours to capture all hours required to complete and support the PR fixes.

TOTAL HOURS PER PR = AVERAGE DIRECT HOURS PER PR * (1+INTEGRATION & TEST FACTOR + SEPM FACTOR)

Apply labor rate to hours.

6.2 METHOD 2: COST PER FIX

If detailed data does not exist to calculate hours per PR. Then a more simplified method can be used. By knowing how much has been spent to date on Software Maintenance, and how many PRs have been closed, and average cost per PR can be derived.

TOTAL AVERAGE COST PER PR = TOTAL ACTUALS FOR SOFTWARE MAINTENANCE / TOTAL PRS CLOSED TO DATE

7 IMPLEMENTATION AND CONTROL

7.1 CONTROL

Once the initial analysis is done, processes need to be put in place to ensure the upkeep of it. Routine updates to the forecast analysis should include rolling in new actuals and fine-tuning assumptions/factors. Forecasts improve over time as more historical data is incorporated and Special Cause Variations are identified and analyzed.

Investigate current processes in place for handling Problem Reports and determine ways to streamline and tighten up the procedures. Is valuable data missing? Is some data unnecessary? What is the flow chart of a Problem Report from Start to Finish? How is the Problem Report enacted contractually, and how is it tracked? Is cost reporting required, and at what level? Asking questions like these, will lead to ways to improve the collection of metrics, and to collect more meaningful metrics.

Inputs into the PR database should be at a level that allows for easy generation of Metric Reports.

PR Database															
PR No	Date Submitted	Product Affected	Versions	Type of Event	Title of Problem	Severity	Priority	Category	Decision	PR Status	Open/ Closed	PR Closed Date	To Be Fixed in Version	Months Aged	Estimated PR Cost

Figure 24: PR Database

Example of Sample metrics/reports:

PRs Opened By Severity

			Quantity Open PRs				
Product	**Version**	**DSLOC**	**Severity 1**	**Sev 2**	**Sev 3**	**Sev 4**	**Sev 5**
Product A	v1.0						
Product A	v1.1						

Figure 25: Quantity of Open PRs by Severity Level

Status of PRs

		Quantity of PRs						Average
Product	**Version**	**Accepted**	**Rejected**	**Deferred**	**Opened**	**Total**	**Backlog**	**Months Aged**
Product A	v1.0							
Product A	v1.1							

Figure 26: Status of PRs

PR Backlog

Product A v1.0	**Sev 1**	**Sev 2**	**Sev 3**	**Sev 4**	**Sev 5**
Quantity Open PRs					
Quantity Closed PRs					
Quantity Backlog PRs					
Current (cum) BMI					
3 mo Ave BMI					

Figure 27: Problem Report Backlog

7.2 Defending your Budget or Estimate

Software Maintenance cost estimates in 1976 ranged from 50 to 75 percent of the overall software life-cycle costs. (Boehm, Software Engineering IEEE Transactions of Computers, 1976). It is common for the cumulative cost of maintenance to exceed the cost of full-scale development by the end of the fifth calendar year of operation. (NCCA/AFCAA, 2008). The weight of maintenance sits heavy on a program's budget; more emphasis should be placed on estimating such an extensive amount of a program's lifecycle.

The previous sections give the tools needed to visually depict and quantify Software Maintenance and what a specific level of funding means (tied to a specific closure rate scenario). They provide a way to address competing demands (affordability and essential maintenance), and determine what makes the most sense for the product being analyzed. The method proposed is based solely on the historical data obtained from the existing product. It aligns with the program's unique lifecycle sustainment profile. It departs from generic high level constructs based on significant assumptions of commonality. It proposes a way to estimate Software Maintenance that is practical, data-driven, adaptable, and defensible.

When asked those hard questions: "What if your budget was cut by X dollars?", "What would be the impact?" a response is now attainable. "An X% budget cut, impacts the Program Office's sustainment portfolio by slowing defect resolution (Y% PRs will go unfixed) and increasing the backlog (or BMI) by Z." Reduced support yields increased risks. It will delay key software patch(es), limiting ongoing support to

the fleet/warfighter/user. Reduced sustainment adversely impacts the ability to rapidly respond to emerging threats and Information Assurance risk reduction. The ability to quantify a dollar impact to Sustainment is vastly beneficial in the existing fiscally-constrained environment. With a PR analysis in place, impact statements can be even more detailed to call out specific PRs that will go unfixed, and how that will impede the warfighter. The "ripple" effect can be shown in the forecast models.

8 LESSONS LEARNED

Understand your product. What does the future hold in store for the product? How long will it be sustained? Are there any known enhancements? Will the prime maintaining activity be changing?

Understand your data. Understand what maintenance is being included within the Problem Reports. Are there other types of Software Maintenance that are not be included?

Continue to fine-tune the analysis and incorporate any needed improvements to the process.

Actual cost and/or hours per PR can be difficult to accurately capture.

Forecast (data out) is only as good as data collection (data in).

The best way to defend a budget is with actual data that allows the quantification of Software Maintenance.

9 REFERENCES

Boehm, B. (1976). *Software Engineering IEEE Transactions of Computers.*

Boehm, B. (1981). *Software Engineering Economics.* Englewood Cliffs, NJ, : Prentice-Hall.

Croll, P. R. (2003, November). Strategies for Applying the CMMI to Software Maintenance .

DAU. (2005). Software Maintenance ReviewV3_DAU.pdf.

IEEE. (2009). *IEEE Std 1044. IEE Std 1044- IEEE Standard Classification for Software Anomalies.*

Institute of Electrical and Electronics Engineers, I. (1998). *IEEE/EIA Standard 12207.0:1996 Industry Implementation of International Standard ISO/IEC12207:1995 — (ISO/IEC 12207) Standard for Information Technology —Software life cycle processes, Institute of Electrical and Electronics Engineers, Inc.* New York, NY.

Kan., S. (2002). *Metrics and models in Software quality engineering, 2nd edition.*

NCCA/AFCAA. (2008). *Naval Center for Cost Analysis, Air Force Cost Analysis Agency. Software Development Cost Estimating Handbook- Volume 1.*

P.Pande, R. R. (2002). *The Six Sigma Way Team Fieldbook, An Implementation Guide for Process Improvement Teams.* New York, NY: McGraw-Hill.

Stuparu, D., Tomita, V., & Stanciu, M. (2010). The Cost Estimating Relationships (CER's)- Modern Method for Predicting Cost. *LFAR*.

Thomas Pyzdek, P. A. (2010). *The Six Sigma Handbook, 3rd Edition.* McGraw-Hill Companies, Inc.