

An Examination of the Effects of Requirements Changes on Software Maintenance Releases

GEORGE STARK^{1*}, PAUL OMAN², ALAN SKILLICORN³, CAPT. RYAN AMEELE⁴

¹ The IBM Corporation, 11400 Burnet Road MD 9236, Austin, TX 78758, U.S.A.

² Software Engineering Test lab, University of Idaho, Moscow, ID 83844-1010, U.S.A.

³ The MITRE Corporation, 1150 Academy Park Loop #212, Colorado Springs, CO 80910, U.S.A.

⁴ SSSG/NDWSE, 1050 E. Stewart Ave, Peterson AFB, CO 80914-2902

SUMMARY

Requirements are the foundation of the software release process. They provide the basis for estimating costs and schedules as well as developing design and testing specifications. Thus, adding to, deleting from, or modifying existing requirements that have been agreed to by both clients and maintainers during the execution of the software maintenance process impacts the cost, schedule, and quality of the resulting product. The basic problem is not with changing requirements per se, the problem is with inadequate approaches for dealing with them in a way that minimizes and communicates the impact to all stakeholders.

Using data collected from one organization on 44 software releases spanning seven products, this paper presents two quantitative techniques for dealing with requirements change in a maintenance environment. First, exploratory data analysis helps to understand the sources, frequency, and types of changes being made. Second, a regression model helps managers communicate the cost and schedule effects of changing requirements to clients and other release stakeholders. These two techniques can help an organization provide a focus for management action during the software maintenance process.

This paper to appear in Journal of Software Maintenance

No. of Figures: 11. No. of Tables: 4. No. of References: 8.

KEY WORDS: requirement volatility, maintenance release management, risk, maintenance productivity, schedule prediction

1. INTRODUCTION

Requirements management involves establishing and maintaining an agreement between the client and supplier, which includes specific information on the technical content, performance and functionality that will be included in the software release. This agreement forms the basis for estimating, planning, performing, and tracking the project's activities. As new requirements are added to the release or existing requirements are deleted or modified from the set, the release cost, schedule, and quality are impacted. These changes to the requirements -- after the basic set has been agreed to by both clients and maintainers -- are known as requirement's volatility.

Requirement's volatility is common in the software industry. According to (Jones, 1994) more than 70% of large applications (i.e., over 1000 function points) experience

requirements volatility. Research by (Standish, 1995) and (Gibbs, 1994) conclude that poor requirements and inadequate risk management contribute to low quality and poor software delivery success rates. The Software Engineering Institute (SEI) believes that organizational processes are a major factor in the predictability and quality of software and requirements management is one of the key attributes in their model of a mature organization (SEI, 1994).

In the maintenance environment, requirements are gathered through change requests that identify feature additions and/or defect corrections. Change Requests are generated from a variety of people including decision-makers, system operators, maintainers, and external interface teams. These people have different backgrounds and levels of understanding associated with computers and system operations. This diversity often leads to misinterpretation of the intent and scope of the change request by the development team resulting in an incomplete or incorrect requirement. This misinterpretation impacts the sizing and effort associated with implementing the requirement. Furthermore, throughout the release process, the requirements often change as a natural consequence of the changing world. During release planning, requirements analysis, design, and test reviews, new priorities are established and changes to the release content are requested in the form of change requests being added and/or deleted from the release. This volatility in the requirements makes it difficult to develop dependable release schedules and budgets. (Curtis, Krasner, and Iscoe, 1988) concluded that accurate problem domain knowledge is critical to the success of a project, and requirements volatility causes major difficulties during development. While these conclusions confirmed most people's intuitions they were not precise enough to allow managers to act on their projects. (Lubars, Potts, and Richter, 1996) went further by interviewing 23 project teams and recommending organizational solutions rather than technological ones to the requirement analysis issue. In no case did they find a coherent relationship between requirements analysis and project planning. This paper explores that relationship by looking at requirement's volatility in a large software system (8 million LOC) spanning multiple languages, environments, and client organizations. In this study we had three primary goals:

- Document the extent and characteristics of changing requirements across 40 releases of large software systems,
- Analyze the requirements volatility data with respect to the origin, timing, and impact of the change requests on the project planning process; and
- Identify economic models and management actions in the planning process that project directors can use to mitigate the negative impacts on software schedule and quality.

Section 2 provides background on the software production environment. In Section 3 we define the data collected and measurements used in this paper. Section 4 presents analysis of the data relative to the frequency, type, and source of requirements volatility experienced in our systems from 1994 to 1998, inclusive. Next, Section 5 shows a preliminary model for predicting schedule change from volatility and effort data derived from individual releases. Finally, in the conclusions we describe some of the benefits

we've achieved by using this model in our environment. We hope these data form a useful basis for other software maintenance managers to make better decisions in the planning and execution of releases that occur in their own software environments.

2. BACKGROUND

The Missile Warning and Space Surveillance Sensors (MWSSS) management office is responsible for maintaining seven products executing in ten locations worldwide. Combined, the products contain more than 8 million source lines of code written in 22 languages. Some of the systems are more than 30 years old and the youngest became operational in 1992. All seven products operate in a hard real-time environment and have a small set of users.

In our maintenance environment a *requirement* is defined as *an approved change request*. The change requests are maintained in a repository and are reviewed for prioritization at a release planning session. During the release planning session, the client and supplier agree to set of requirements and document their agreement in a version content notice (VCN). The VCN is then used as the basis for developing a project schedule and cost estimate for the release by the development team. This team then proposes a plan of record to the Configuration Control Board for concurrence. At this time, progress against and changes to that plan are monitored for corrective action, if necessary. For more information on the MWSSS and its software management, see (Stark, 1997).

3. DATA COLLECTED AND DEFINED MEASURES

Because requirements management is a primary factor in our success, we collect data on the

1. type of requirement and detailed need
2. planned and actual effort-days for each requirement
3. planned and actual number of calendar days for a version
4. requirements changes made to the version after plan approval (i.e., type of change & requesting group)

Table 1 shows the taxonomy of requirement types and detailed categories. The planned and actual effort-days associated with each requirement are also maintained. This data is collected by the maintainer's accounting system using the project charge codes assigned to the requirement. The planned and actual calendar days for a version is measured from date of the approval of a plan of record until the date that the release is accepted by the operators in the field. Requirement change types are:

1. added – one placed into the VCN after the initial plan of record is agreed upon
2. deleted – one removed from the VCN after the initial plan of record is agreed upon
3. scope change – one in which the original intent was determined to be incorrect after the initial plan of record is agreed upon.

Table 1. Software Requirement Types and Detailed Needs

Requirement Type	Detail Category
Computational	Incorrect Operand in Equation Incorrect Use of Parentheses Incorrect/Inaccurate Equation Rounding or Truncation Error
Logic	Incorrect Operand in Logical Expression Logic Out of Sequence Wrong Variable Being Checked Missing Logic or Condition Test Loop Iterated Incorrect Number of Times
Input	Incorrect Format Input Read from Incorrect Location End-of-File Missing or Encountered Prematurely
Data Handling	Data File Not Available Data Referenced Out-of-Bounds Data Initialization Variable Used as Flag or Index Not Set Properly Data Not Properly Defined/Dimensioned Subscripting Error
Output	Data Written to Different Location Incorrect Format Incomplete or Missing Output Output Garbled or Misleading
Interface	Software/Hardware Interface Software/User Interface Software/Database Interface Software/Software Interface
Operations	COTS/GOTS SW Change Configuration Control
Performance	Time Limit Exceeded Storage Limit Exceeded Code or Design Inefficient Network Efficiency
Specification	System/System Interface Specification Incorrect/Inadequate Functional Specification Incorrect/Inadequate User Manual/Training Inadequate
Improvement	Improve Existing Function Improve Interface

For each requirement change, we also collected data on which stakeholder group (i.e., user management, site analysts, acquisition management, development contractors) requested the change and for some releases, when in the project the change occurred.

Based on this data we calculated the Planned schedule percentage, requirement volatility, and risk. These measures are defined as follows:

$$\text{PlannedSchedulePercent} = \{1 + ((\text{actualdays} - \text{planneddays}) / (\text{planneddays}))\} * 100 \quad (1)$$

Thus, a value of 100 means the original plan was met (actual days = planned days), while a value greater than 100 means the release was late (actual days > planned days) and a value less than 100 means the release was delivered early (actual days < planned days).

$$\text{RequirementVolatility} = (\text{added} + \text{deleted} + \text{changed}) / (\text{\#requirements in VCN}) * 100 \quad (2)$$

Requirement volatility represents the total change traffic applied to an agreed-to release plan. It can be greater than 100% if more changes occurred than the number of requirements originally planned for the release.

$$\text{Risk} = (\text{\#requirements delivered by type}) / (\text{staff days charged to requirement type}) \quad (3)$$

Risk represents the average productivity for a type of requirements in the taxonomy. Large numbers indicate an easier type (less risk) than smaller numbers.

4. ANALYSIS

This section uses exploratory data analysis and regression analysis techniques to answer the following questions related to our organization's requirement's volatility:

- How much volatility do our product releases experience?
- What kind of changes are most commonly requested?
- When in the release cycle do requirements changes occur?
- Who requests requirements changes?
- How much effort is associated with implementing each requirement type?
- What schedule impact will a requirements change have?

4.1 How much volatility do our product releases experience? What kind?

A total of 123 requirements changes were made to the forty-four software releases under study. Figure 1 is a stacked bar chart showing releases that experienced requirements volatility. Sixteen of the deliveries (36%) had no requirements change; of these, seven were made on or ahead of schedule, four more were within 15% of the original scheduled date, and five were more than 15% late. The remaining twenty-eight releases (64%) had requirements changes, with nine of them having greater than 50% volatility. Table 2 shows the breakdown of the twenty-eight releases experiencing requirements volatility by type of change.

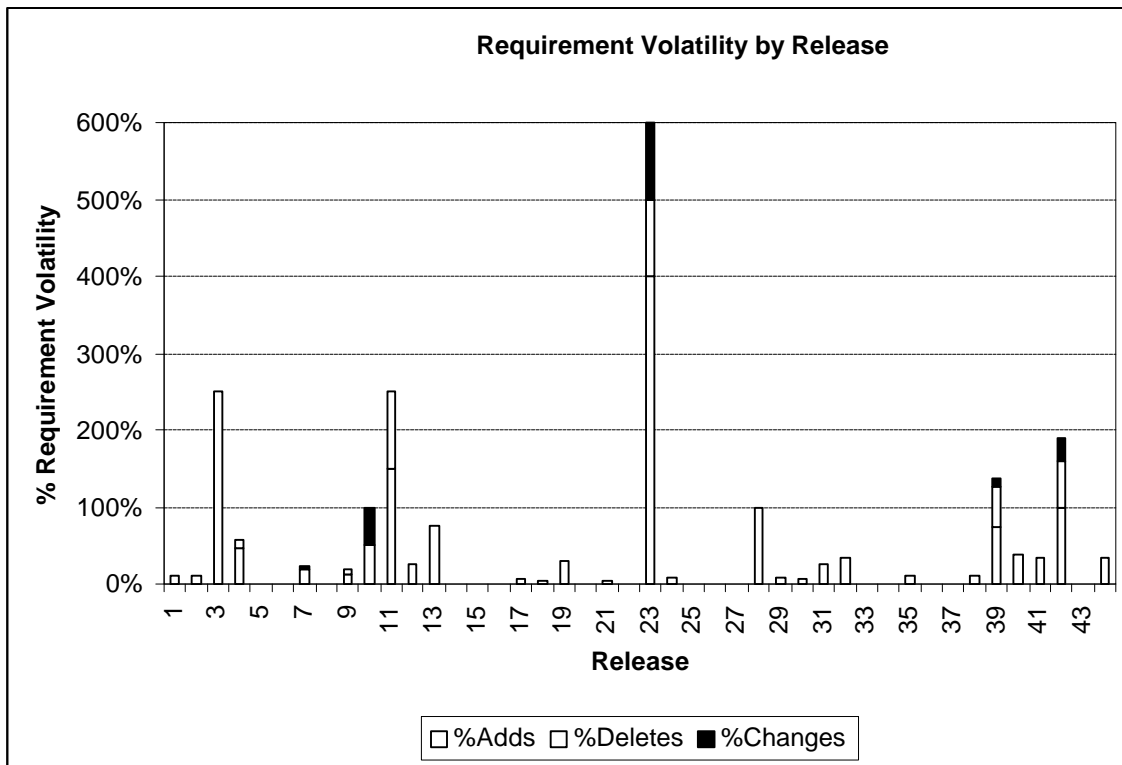


Figure 1. Requirement Volatility by Maintenance Release

Table 2. Breakdown of Releases with Requirements Volatility by Type

Releases with Only Requirements Added	Releases with Only Requirements Deleted	Releases with Only Scope Changes	Released with a Combination of Added, Deleted, and Scope Change
12	8	0	8

The average requirements volatility for these 44 releases is 48% which is significantly greater than the 35% for 60 projects cited in (Jones, 1994). Taken literally, the data from these two studies of multiple projects shows that at least 1/3 and potentially as many as 1/2 of the delivered requirements implemented during system maintenance were not part of the original plan. The largest observed volatility is an astonishing 600%. Additionally, 250% was observed twice. In the case of release 23, the original plan contained only one new feature, this feature was deleted from the release and four new features were added after design work had begun. During the critical design review, one of the new features was changed from a single user modification to a multi-user function. In combination, the result of these changes was a 78% increase in schedule duration. In the case of release 3, a new, high-priority mission was defined during the release and those requirements were added to the version. For release 11, both of the original requirements were deleted from the release because of a configuration change in an interfacing system.

Three new requirements were incorporated as a part of the version content after design work had begun.

Note the releases in Figure 1 are in chronological order with a four year span between release 1 and release 44. Although, recent releases have experienced volatility, the requirement's management process has improved over time as evidenced by the decrease in average volatility. The volatility of the releases in the first two years was 64% (release 1 through release 23) and the average volatility of the last two years was 30% (release 24 through release 44).

Figure 2 shows the distribution of these changes by category. Additions to the release functionality are the most common form of change, followed by deletions, with scope changes occurring least frequently.

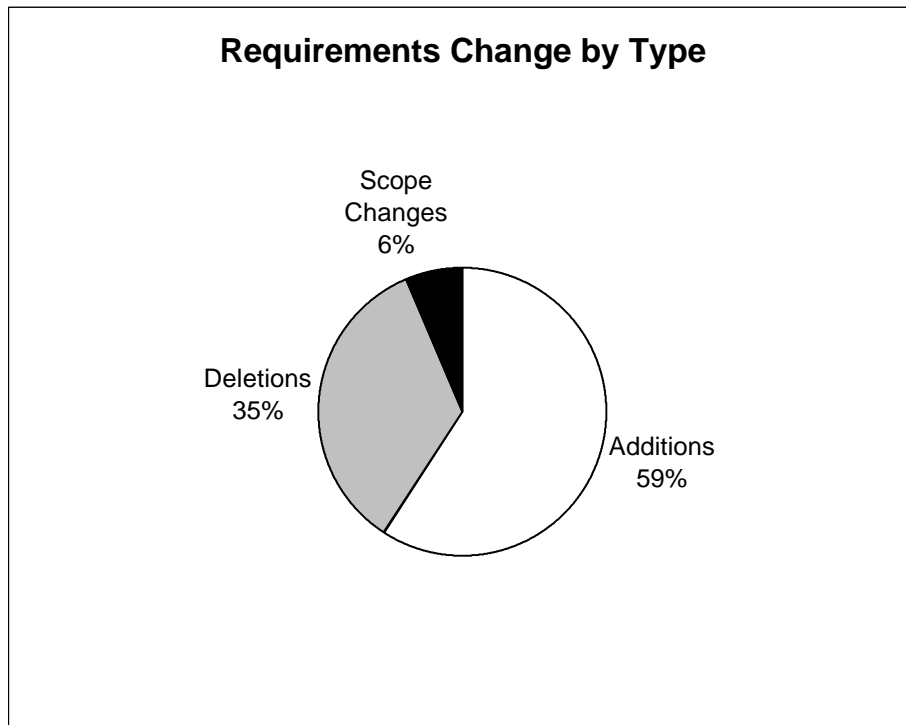


Figure 2. Distribution of Requirement Changes by Type

4.2 When in the release cycle do requirements changes occur?

Because this data was not collected consistently for all releases, this section is based on one release that was approved with 17 requirements to be completed on a 9-month schedule. Figure 3 displays the changes in these requirements over the scheduled time for the release. The requirement changes were processed both formally (through the Configuration Control Board) and informally (agreement between users and maintainers). This chart shows that 20 total changes were made to the release content (volatility of

117%) in the 14 months since project plan approval. This change rate of 1.4 changes/month (about 8% per month) is extremely high according to (Jones, 1994).

The two spikes in February and October occurred after design reviews with stakeholders, when major scope changes occurred to some of the requirements. Nine of the changes occurred in the last five months of the effort and only six of the delivered requirements were a part of the original approved plan.

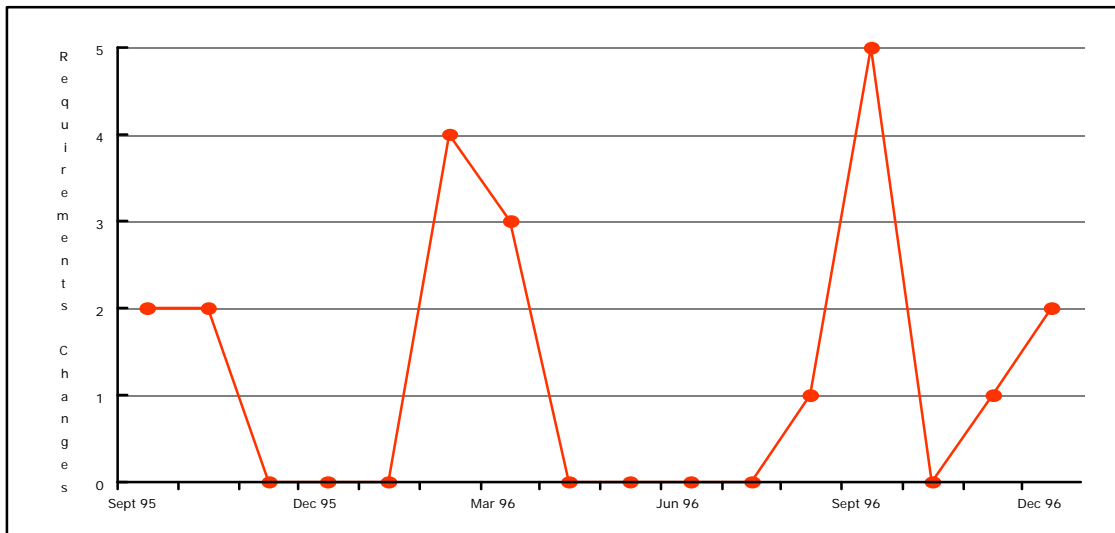


Figure 3. Requirement Changes by Month for One Project

4.3 Who requests requirements changes?

Four groups of people contribute to the maintenance release process: the contractor development team, the acquisition management team, user management personnel, and individual site analysts. Each group contributes to the requirements changes associated with each software product release. Figure 4 shows the percent of changes requested by each group. It is important to notice that the distribution of requirements changes is about 50/50, implying that both the maintainer and client are just as likely to request a requirements change during a release.

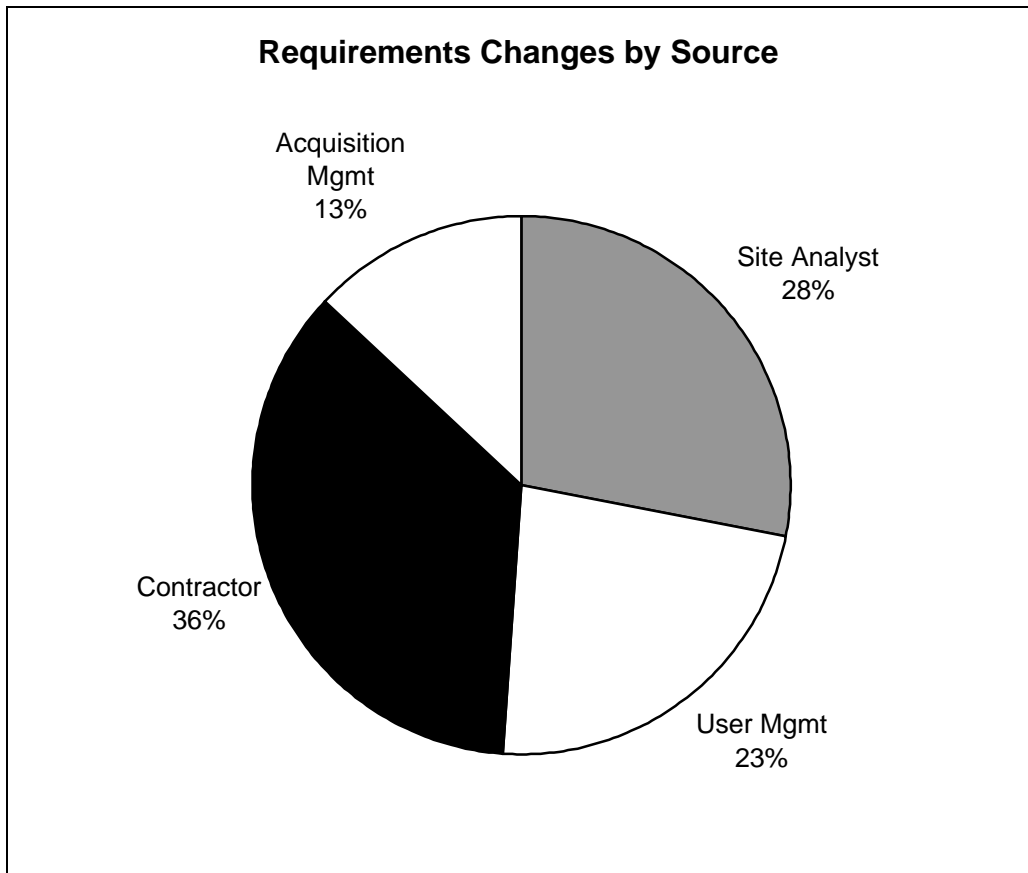


Figure 4. Distribution of Requirements Changes by Source

4.4 How much effort is associated with implementing each requirement type?

We analyzed and categorized 243 requirements (104 feature changes and 139 defect corrections) from eight of the releases to the requirement types identified in Table 1. Figure 5 is a Pareto diagram of this change data. The left vertical axis shows the actual number of changes attributed to each type; the right vertical axis represents the cumulative percentage of requirements and is a convenient scale from which to read the line graph. The line graph connects the cumulative percents (and counts) at each category.

The Pareto diagram indicates that Logic requirements are the most common software change (45 changes or 19% of the total) and data input requirements are the least common (2 changes out of 243). Although it is not shown in Figure 5, the most frequent detailed category is a missing logic or condition test for error handling. Using this information, we have expanded our design and code reviews to specifically look for these logic problems.

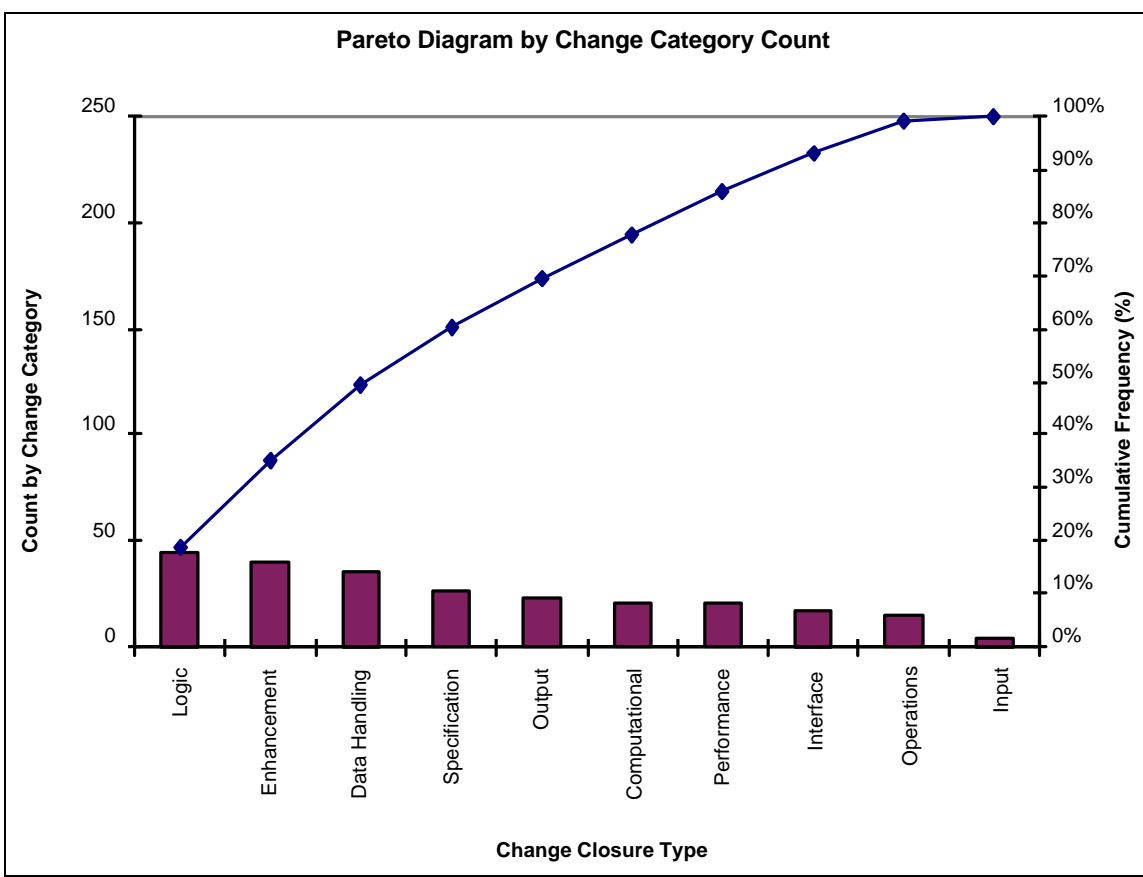


Figure 5. Frequency of Requirements by Type

Figure 6 is another Pareto diagram, but this time it shows the effort required to design, code, and unit test each change. Comparing Figure 5 and Figure 6 demonstrates the importance of requirements change. Although specification changes ranked fourth in the number of changes (26 changes shown in Figure 5), they account for 20% of the total maintenance effort (591 staff-days shown in Figure 6). In contrast, logic changes, which are the most prevalent, are the sixth category in terms of effort?

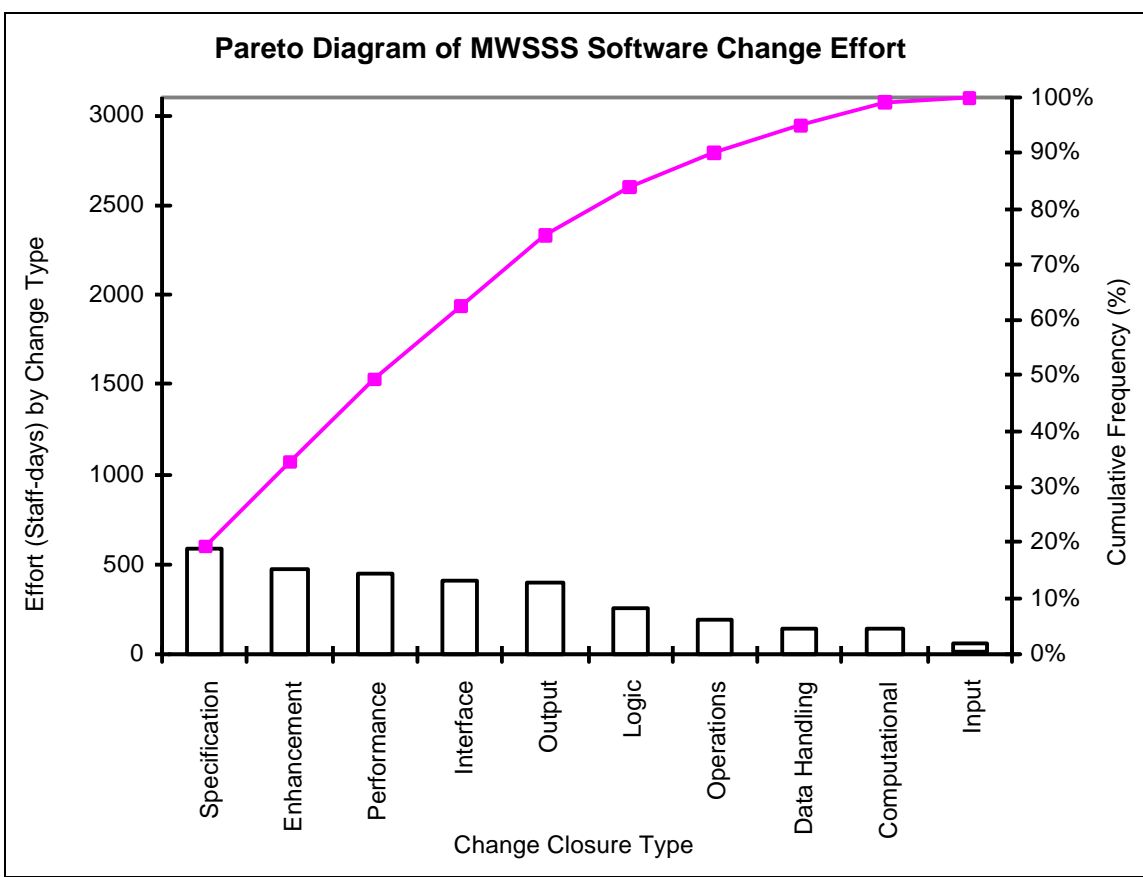


Figure 6. Total Effort by Requirement Type

4.5 What schedule impact will a requirement's change have?

Because we were experiencing so much requirements' volatility and missing scheduled delivery dates, we tried to forecast the schedule impact of requirements volatility. A random sample of 20 releases was chosen to understand the relationship. The initial attempt to graph the percent of planned schedule vs. requirement volatility, as shown in Figure 7, indicates that some sort of transformation is required. Because of the wide variation in the requirement volatility and because we are using percentages, the square root transformation is an obvious candidate (Weisberg, 1980). The square-root transformation spreads out the numbers close to zero and condenses the numbers greater than one making for a linear relationship. The scatter plot with the transformed variable is shown in Figure 8, suggesting that there is a linear relationship in that scale.

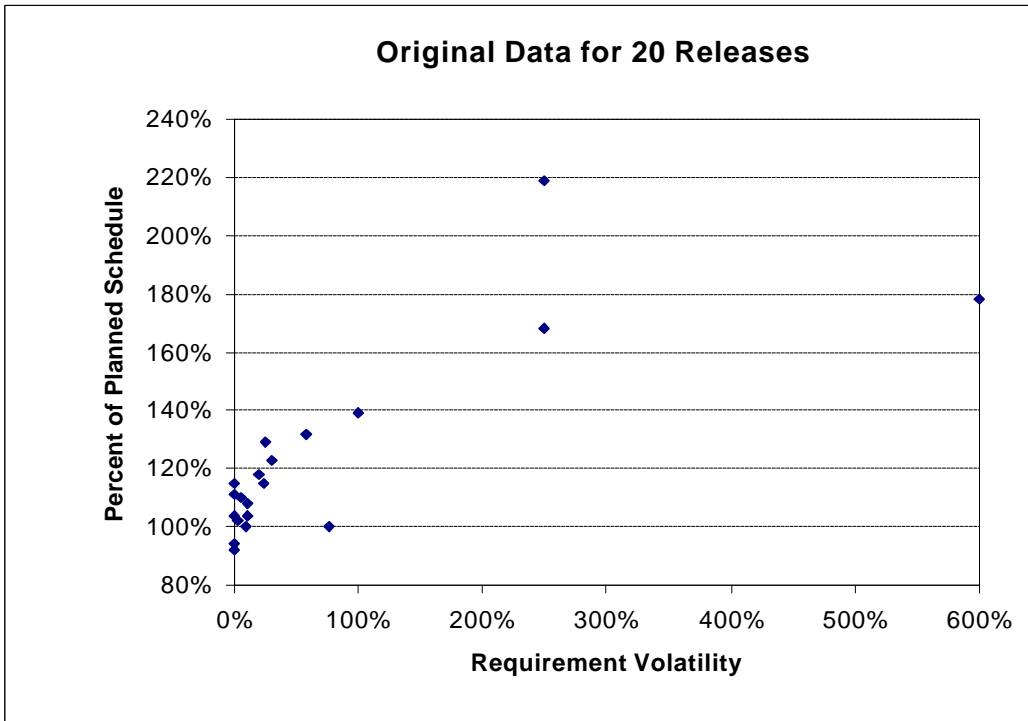


Figure 7. Original Plot of Percent of Planned Schedule vs Requirement Volatility

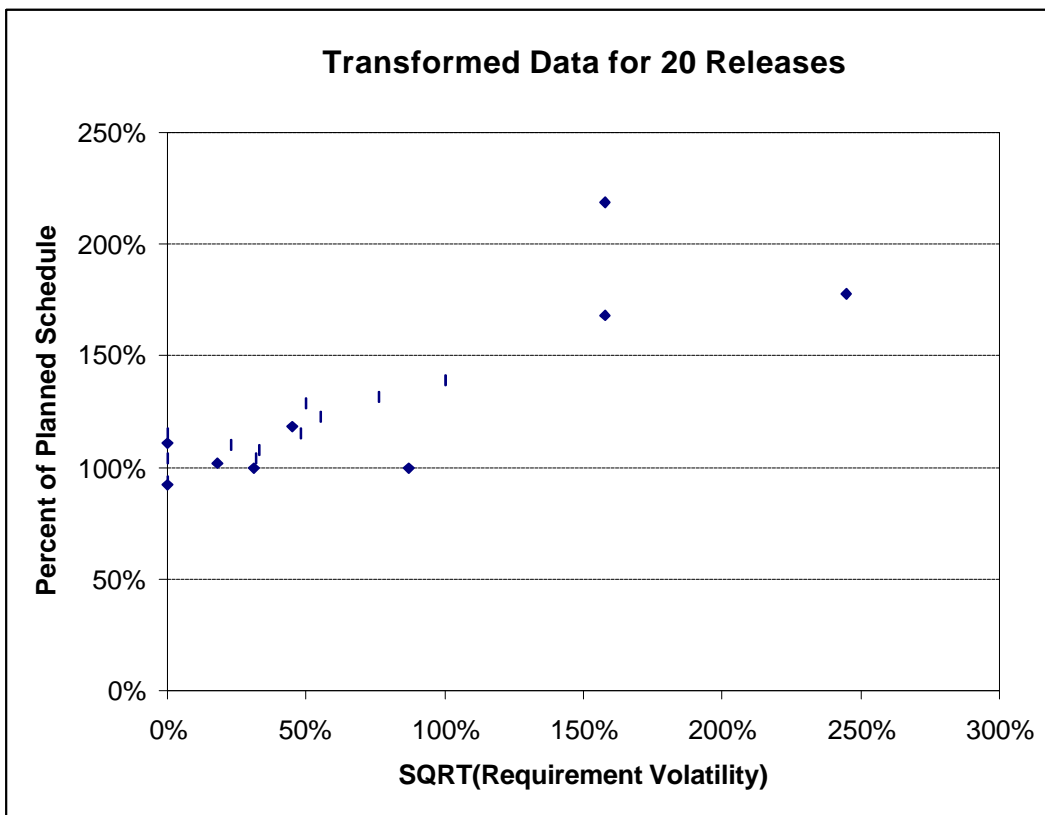


Figure 8. Transformed Plot of Percent Planned Schedule vs. (Requirement Volatility)^{1/2}

Another variable that contributes to the schedule is the risk associated with a new requirement. The units on risk are requirements/staff-day which makes it a measure of productivity for the release. Thus, changes requiring a higher level of productivity to meet schedule pose a greater risk to the release than easier changes. Figure 9 shows the risk vs. percent of planned schedule for the twenty releases in the sample. Clearly, risk has little individual correlation with percent of schedule achieved.

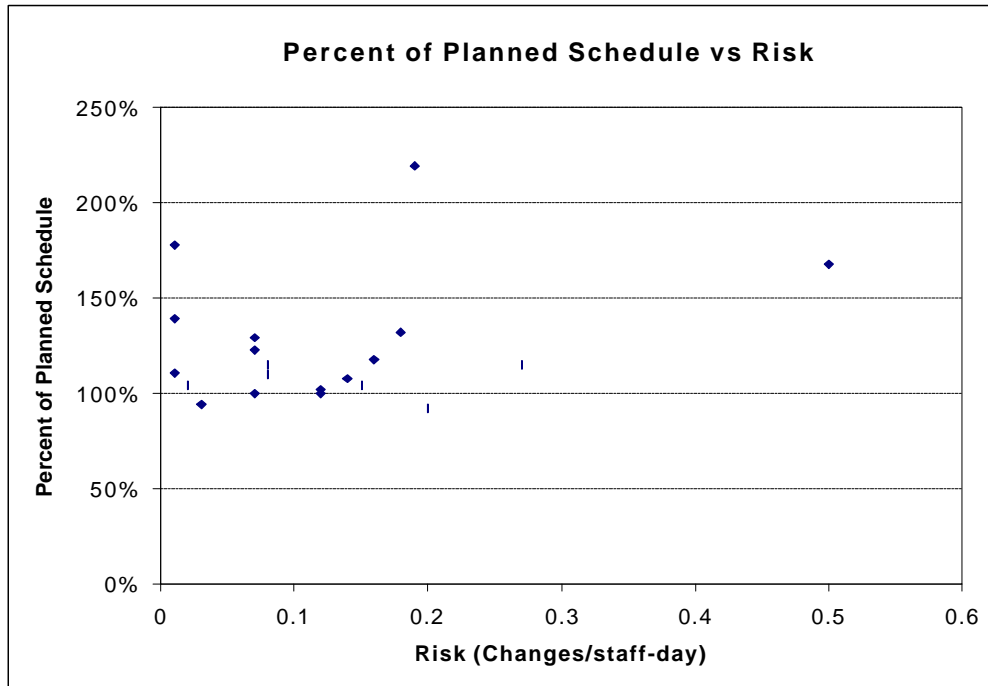


Figure 9. Percent of Planned Schedule vs. Risk

Table 3 summarizes the data from the 20 releases depicted in figures 7-9.

Table 3. Summary of Data Used to Develop Expression (4)

Version	Percent of Planned Schedule	SQRT(Percent of Rqmts Change)	Risk(Change Requests per Staff-day)
1	108	33	0.14
2	104	32	0.15
3	168	158	0.50
4	132	76	0.18
5	115	0	0.08
6	115	48	0.27
7	118	45	0.16
8	139	100	0.01
9	219	158	0.19
10	129	50	0.07
11	100	87	0.07
12	111	0	0.01
13	102	18	0.12
14	123	55	0.07
15	92	0	0.20

16	178	245	0.01
17	104	0	0.02
18	110	23	0.08
19	100	31	0.12
20	94	0	0.03

We performed a linear regression analysis on the risk (see expression (3)) and the square root of the requirements volatility (see expression (2)) using the Microsoft Excel™ Spreadsheet Package. Expression (4) shows the resulting likelihood function.

$$\text{PercentPlannedSchedule} = 0.97 + 0.41\text{RequirementsVolatility}^{1/2} + 0.23\text{Risk} \quad (4)$$

The proportion of variance explained by this model (R^2) is 0.72 and the standard error of the estimate is 0.17. Table 4 shows the analysis of variance for this regression equation. Since $F = 21.68$ exceeds $F(0.05, 2, 17) = 6.11$ we can conclude that the independent variables are related to the percent of planned schedule.

Table 4. Analysis of Variance for Regression Results of Expression (4)

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	2.00	1.39	0.70	21.68	0.00
Residual	17.00	0.54	0.03		
Total	19.00	1.94			

To further examine the model for reasonableness we plotted the standardized residuals against the predicted values. If the fitted model does not give a set of residuals that seem reasonable, then some aspect of the model will be called into doubt. The scatter plot of the residuals from the fitted regression is given in Figure 10. Notice that 95% of the residuals fall in the range +2 to -2, and only one point falls outside the range +3 to -3. Furthermore, the plot exhibits no systematic trend. The combination of residual attributes adds confidence to the model.

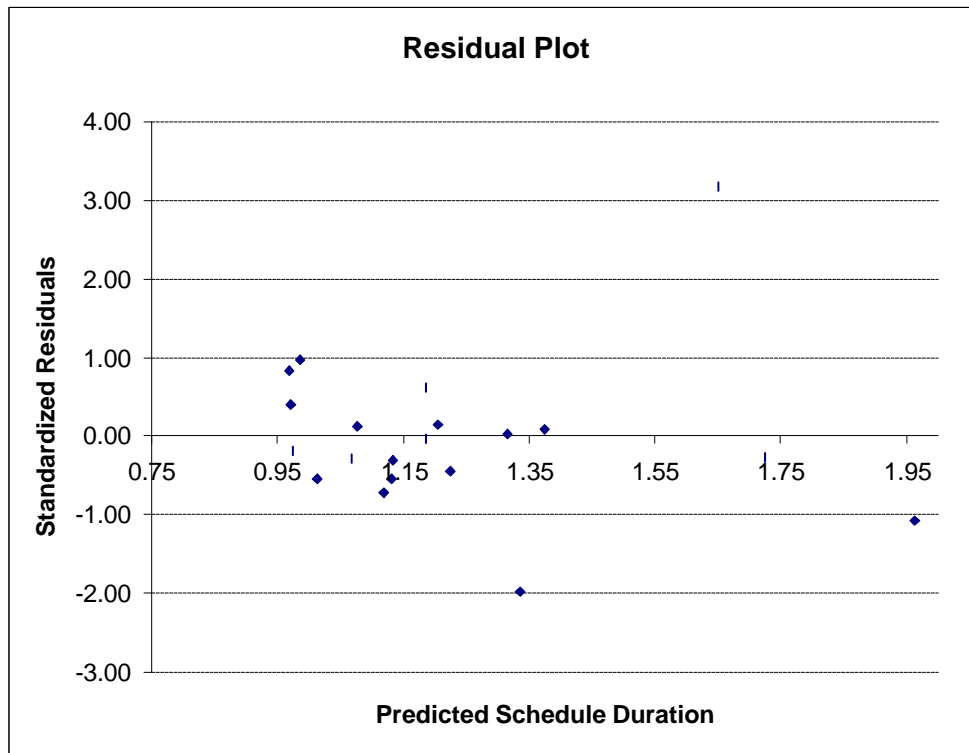


Figure 10. Standardized Residuals vs Predicted Schedule Duration

Using expression (4), a release that experiences no requirement's changes and hence no change in expected risk should be delivered in 97% of the planned scheduled time. Notice that the schedule change goes up regardless of whether the requirements change was an addition, deletion, or scope change because the input to the model is percent of requirements change. This makes sense, since in a maintenance environment, removing planned requirements from a release often involves removing code and always requires effort to change the design and user documentation as well as changes to the test environment.

Figure 11 shows the results of applying the model to all 44 releases. The line $y = x$ in Figure 11 indicates the ideal situation where the predicted value using expression (4) would be equal to the actual schedule duration on our projects. Comparing the data points with the line, we see that the model performs much better in the 115% to 130% of planned schedule range and becomes more optimistic as the predictions get larger (i.e., > 150% of plan). This may indicate the need for another independent variable (such as timing of change) to account for major changes going into a release.

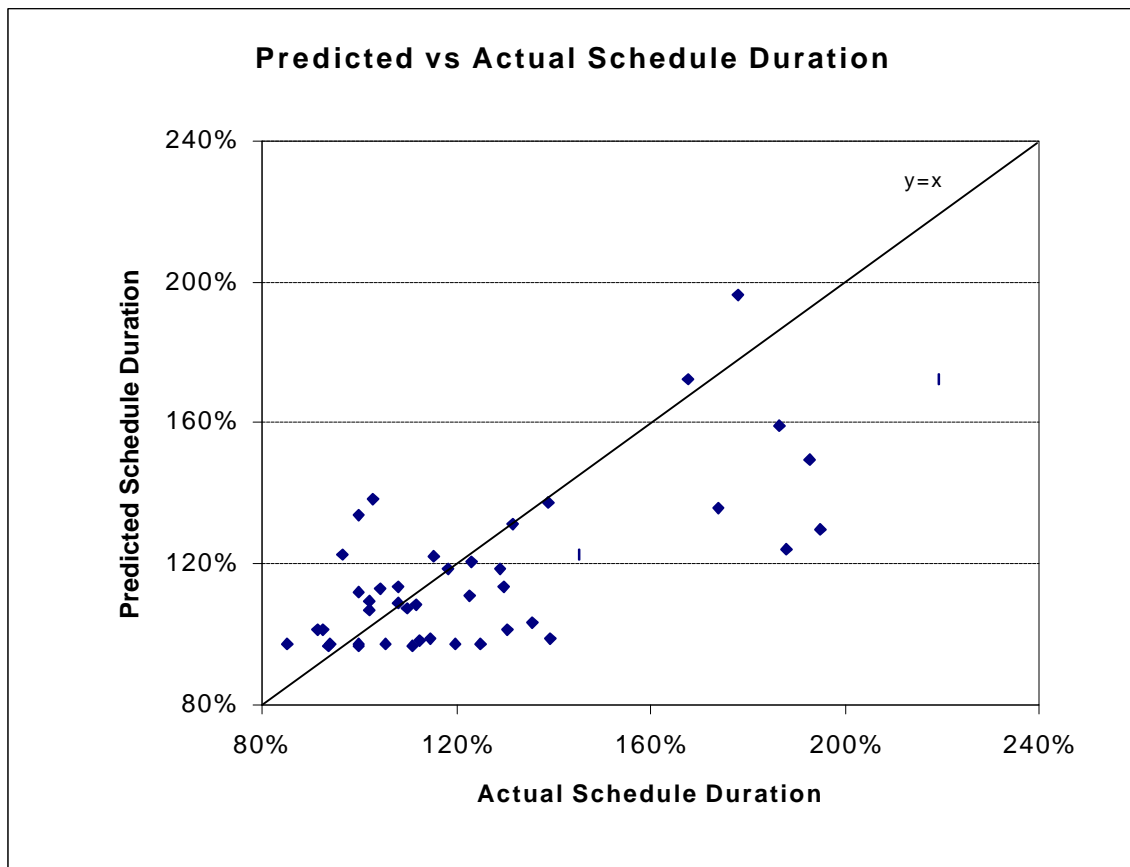


Figure 11. Predicted vs Actual Schedule Duration Compared with the Line $Y=x$

5. RESULTS

Several improvements are underway because of this analysis. First, we recognized that in our maintenance environment requirement prioritization and sizing are problem areas. If we improve the prioritization of requirements during release planning we should be able to reduce the number of releases with added requirements. Along the same lines, deletions should drop if more accurate requirement sizing can be done to ensure that releases are executable in the schedule allowed. We do believe, however, that our process for eliciting the intention of the requirement is working as evidenced by only 10% of releases having scope changes. Thus, we have instituted the development of a program called Mystic to help facilitate prioritization meetings and project planning.

A second improvement is the use of the requirement taxonomy for sizing. By reviewing requirements and accurately assigning them to the taxonomy, managers and engineers can estimate the staff-days required to design, code, and test individual changes. For example, the average effort for interface requirements are 24 staff-days with a standard deviation of 50 staff-days, while the average effort due to functional specification changes is 23 days with a standard deviation of 29 staff-days. (Note the non-normal distribution of the effort data is a natural consequence of the task. Some changes are

much, much harder than others skewing the distribution). Of course, this estimation model is continually updated by comparing the actual effort against the estimated effort, and updating the taxonomy and cost information as each release is completed. While the current information is highly variable for each detailed category, we expect the effort data to converge around a reasonable mean as we collect more data. If the convergence does not occur, we plan to convert to order statistics and percentiles for the estimation process. This will increase our confidence in the estimates. Sudden changes could indicate a need to reexamine our processes or a need to change the staff implementing the requirement. Even with the current variability, we believe using the historical data is the best method for estimating individual change effort.

Third, by understanding the source and timing of requirements changes we have identified corrective actions related to design and code reviews. For example, if the client is requesting changes at a review, then we can stop the process and call a special Joint Application Design (JAD) session or request a prototype of the requirement as a means for controlling the problem. Also, the release management process was changed to include more detailed requirements tracking and schedule management after release 23. The changes included a more disciplined review of requested changes through bi-monthly status meetings. The review now includes a requirements analysis checklist that covers items such as the mission need, technical need, and cost/schedule impact to help the project manager control the baseline.

Fourth, the regression model has been used to facilitate communication among the project stakeholders when discussing requirements changes. For example, one release contained 15 planned requirements scheduled for delivery in 91 calendar days--the client wanted to drop two of the requirements and change the scope of a third at preliminary design. Managers estimated the change in risk to version delivery to change from 0.14 (15 changes in 108 staff-days) to 0.1 (13 changes in 130 staff-days). Using the model, managers forecast the overall schedule impact to be $[0.97 + 0.41*(0.2)^{1/2} + 0.23*(0.1)] = 1.18$ or an 18% schedule slip. An 18% slip is equivalent to 16 days added to the 91-day schedule. These 16 days would have cost the client an additional \$60,000. During discussion about the model and the prediction, the client decided that this schedule slip was not acceptable to the overall mission of the release; therefore, the client decided not to pursue the changes, but to incorporate the scope change in the next release. The metrics-based model facilitated objective communication with the client concerning version release plans and status.

6. CONCLUSIONS

Adding new requirements, deleting partially implemented requirements or significantly modifying existing requirements after the basic set of requirements have been agreed upon, is a common problem in software maintenance. Unfortunately, there are few good methods, tools, or approaches to dealing with the problem today and precious little quantitative evidence to help develop theories. Based on our analysis we can make the following conclusions:

-
- Additions are the most common form of requirements change with the average volatility being 48% for projects in our environment.
 - Clients and maintainers change requirements almost equally, but different techniques are necessary for controlling the two groups. JAD and prototypes are useful for dealing with clients, while strong discipline and more detailed review items are needed for maintainers.
 - Most requirements change occurs during or immediately following a formal review. Informal reviews and desk-checking do not garner the same attention leading to changes.
 - Logic changes for error handling and condition tests are the most common type of maintenance requirement in our environment, but new feature development takes the most effort.
 - An “economic impact” model can be used as a basis (within limits) for discussion between clients and suppliers when deciding to change release requirements.
 - An accurate measurement program is useful for preventing and controlling requirement’s volatility.

Acknowledgements

We acknowledge Karen Fox for her effort in maintaining the measurement repository and the anonymous reviewers for helpful comments on a poor first draft of this paper. Finally, we thank Ned Chapin for his guidance, determination, and patience in revising this paper.

References

- Curtis B, Krasner H, Iscoe N. 1988. A field study of the software design process for large systems. *Communications of the ACM* **31**(11):1268–1287.
- Gibbs W. 1994. Software’s chronic crisis. *Scientific American* 271(3):86-95.
- Jones C. 1994. *Assessment and Control of Software Risks*; Prentice-Hall International: Englewood Cliffs NJ; pp. 93–98.
- Lubars M, Potts C, Richter C. 1996. A review of the state of the practice in requirements modeling. In *Proceedings International Symposium on Requirements Engineering*; IEEE Computer Society, Los Alamitos, CA, pp. 2–14.
- SEI. 1994. *Software Process Maturity Questionnaire Capability Maturity Model, version 1.1*; CMU/SEI-94-007, Software Engineering Institute, Carnegie Mellon University: Pittsburgh PA; 57 pages.
- Standish. 1995. *The Scope of Software Development Project Failures*; The Standish

Group: Dennis MA; <http://www.standishgroup.com/chaos.html>, 10 pages.

Stark G. 1997. Measurements for managing software maintenance. *CrossTalk* 10(7):14-21.
<http://www.stsc.hill.af.mil/CrossTalk/1997/jul/maintenance.asp>

Weisberg S. 1980. *Applied Linear Regression*; JohnWiley & Sons, Inc.: New York NY; 283 pages.
