# Software Maintenance Management Strategies: Observations from the Field

**George Stark, MITRE**

**Paul Oman, Univ of Idaho**

## Abstract

There is much literature describing software maintenance process models, but few comparative studies on the approaches used by managers in the field. This paper describes three software maintenance management strategies currently implemented by six organizations. The strategies are compared on the attributes of performance, economics, efficiency, and end-product quality. The paper defines measurements for each attribute and describes the data collected over the past two years. Our observation is that each strategy has attributes that make it appealing for implementation by a software maintenance project manager. The key task for a manager is defining the attribute that they would most like to optimize and choosing a strategy that supports that goal.

## Introduction

According to IEEE Std 1219, "IEEE Standard for Software Maintenance," software maintenance is

> *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.*[1].

The importance of software maintenance in today's industry cannot be overestimated. It is widely recognized as the highest cost phase of the software life cycle with estimated costs of between 60% and 80% of the total software budget [2-4]. Given this high cost, some organizations are looking at their maintenance process as an area for competitive advantage [5].

Several authors have noted that maintenance of software systems intended for a long operational life pose special management problems [6-8]. The Software Engineering Institute (SEI) believes that organizational processes are a major factor in the predictability of cost and quality of software [9]. Furthermore, Alexander and Davis [15] point out that today's project managers choose a software process model using ad hoc, unjustified, and undocumented criteria. The selection of an inappropriate process model can result in (1) a release that does not satisfy user needs (reflected by the number of user-created problem reports and requirements volatility), (2) missed schedules with increased cost, and (3) more defects being released to the field.

At a macro level, two distinct activities occur during software maintenance: Requirements Validation and Maintenance Release. The requirement validation process consists of reviewing individual change requests for impact. A change request can be either a defect correction or an enhancement to the system. An engineer reviews the contents of the change request to ensure clarity, completeness, feasibility, and consistency with outstanding change requests. Once the change request is determined a valid requirement, it is placed in a queue awaiting implementation during a maintenance release. Most organizations have a backlog of change requests available. While the requirement validation activity is central to the software maintenance process, it is not a focus of this paper.

Early published software maintenance release models viewed the implementation of changes from the point of view of the software engineer dealing with an individual change [10-13]. Even more recent release descriptions [1, 14] consider software maintenance as a one change at a time function. In our experience, however, a maintenance release is generally defined as a group of change requests based on the user priority, changes that affect the same subsystem, logical timing, and other activities underway at the user location. Under this scenario, a management view is needed to understand the cost, quality, and timeliness implications of different implementation strategies. We identified four release implementation strategies available to software maintenance managers:

- Fixed staff/variable schedule
- Fixed schedule/variable staff
- Variable schedule/variable staff
- Fixed staff/fixed schedule

The overall objectives of this paper are: (1) to present software maintenance managers with the options available to them for implementing software releases, (2) to report our observations of these options in terms of four process criteria, and (3) to draw conclusions based on the first two objectives. Section 2 of this paper describes three of the four management strategies. The fixed staff/fixed schedule strategy was not implemented by any of the six organizations that we observed, thus it is not described further. Section 3 identifies the key strategy criteria considered in this study (i.e., performance, economics, efficiency, and quality) and discusses the data collected. Section 4 presents the results and discusses their implications. Finally, Section 5 summarizes our conclusions and points out the limitations of our study.

## 2. Maintenance Management Options

This section describes three management strategies for software maintenance organizations that reside several hundred miles from their users. According to Pigoski, distance from the user has distinct advantages and

disadvantages [4]. The advantages include: better documentation, formal release procedures, and more clearly defined processes. The disadvantages are: significant training is required, user support can suffer, morale can decrease if the wrong people are assigned. We observed all of these effects on the programs we reviewed. Additionally, we observed that a strong measurement program supported the management throughout the maintenance process. Measurement helped managers to communicate more clearly with their customers on issues such as software cost estimates, requirements volatility, schedule constraints, and product quality analysis.

The strategies are labeled as: (1) fixed staff/variable schedule, (2) fixed schedule/variable staff, and (3) variable schedule/variable staff. A fixed staff/fixed schedule strategy was not implemented by any of the six organizations that we observed, thus it is not described further. For each strategy it is assumed that the requirement validation activity is completed and the change request is available for incorporation in a version release. It is from this point in the maintenance life-cycle that the three strategies are compared.

**Fixed Staff/Variable Schedule Strategy**

Figure 1 shows the implementation of the fixed staff/variable schedule strategy. In this implementation, there is a fixed pool of maintenance engineers available to the organization. A maintenance engineer takes a change request off the backlog queue and begins understanding, designing, coding, and unit testing the change. The available staff work as many changes as possible through unit test until a “release freeze” is declared by management. The release freeze is scheduled depending on the operational mission needs of the system, but is typically every six months for the systems we observed. Integration test time is scheduled on the operational platform for 30 days after the release freeze. All changes that the engineers declare complete through unit test are then bundled into a release for integration test and formal turnover to the configuration management team. Note there in nothing inherent in the process that precludes code inspections or peer reviews, and some engineers performed them. They were not, however, formalized in the organizations that we observed.

Since the content is determined by the changes the engineers have unit tested up to the release freeze date, and the release schedule is determined by the integration test schedule, this process, should always contain 100% of the unit-tested content and always be delivered on-time. The advantage of this process over the other two is its flexibility in choosing release content and ease with which work is assigned.

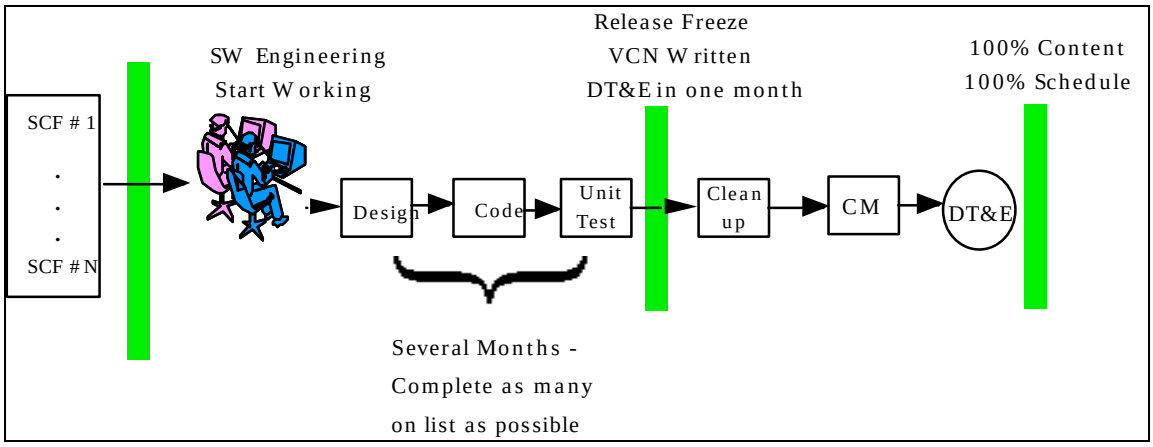


**Figure 1. Fixed Staff/Variable Schedule Software Process**

## Variable Staff/Fixed Schedule Strategy

Figure 2 depicts the flow of the variable staff/fixed schedule strategy. In this process, there may be a fixed pool of maintenance engineers available, but they are allocated to individual software releases. The product release date is established between the customer and supplier based on a mission need date. Once the date is established and the "high priority" changes are agreed to for the mission, the system analysts prepare a preliminary version content notice (VCN) and a release plan. The preliminary VCN may contain additional changes to the software based on the change request backlog or the schedule. The software engineers begin working on the priority changes while the preliminary VCN and plan are being reviewed by management. If the engineers or managers feel that the VCN is too ambitious or that some changes do not make sense within the context of the version, the content or staffing may be re-negotiated. A final VCN for the release is then issued prior to the completion of all unit testing. After unit test the new version is delivered to the configuration management team for integration and operational test prior to formal acceptance by the user.

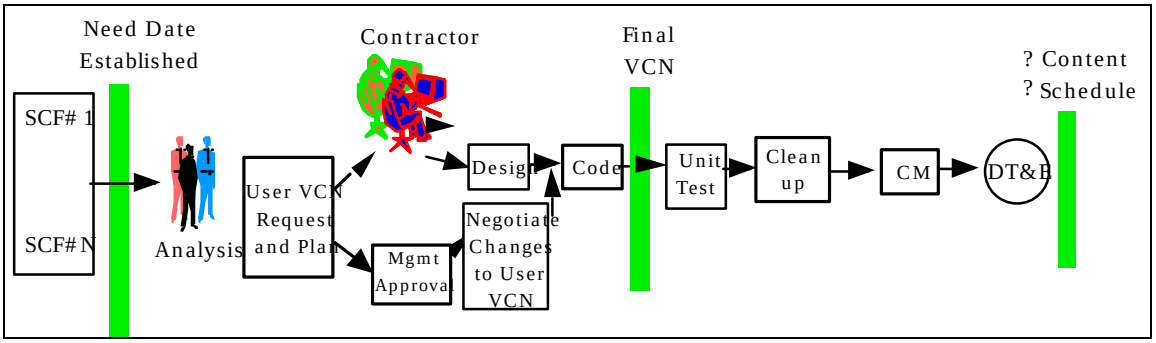


**Figure 2. Variable Staff/Fixed Schedule Process**

Since the planned schedule is established between six and eighteen months in advance for this process and some content is negotiable throughout the

process, the actual content and delivery date vary with each release. The advantage this process has over the other two is its clearly defined need dates and priority changes. This forces the maintenance team to actively coordinate the release requirements with the user community and focuses the release on achieving the mission goals.

### Variable Staff/Variable Schedule Strategy

The variable staff/variable schedule strategy is shown in Figure 3. In this strategy, there is not a fixed staff of maintenance engineers, the staff size and skill varies with each release. To determine the release content system users submit a VCN request to the maintenance team. The maintenance team reviews the request and estimates the cost, schedule, and risk associated with the release.

Based on this review, the maintenance team may either add content or negotiate less content with the users. After the content is determined, the maintenance team negotiates a release cost and schedule with a software maintenance contractor to design, code, test, and integrate the release. Milestone reviews are held during the implementation and changes to the plan are made as necessary. The three technical reviews that occur in this process, but are not explicit in the other two processes, are an advantage because the reviews identify issues and help to ensure quality in the release.

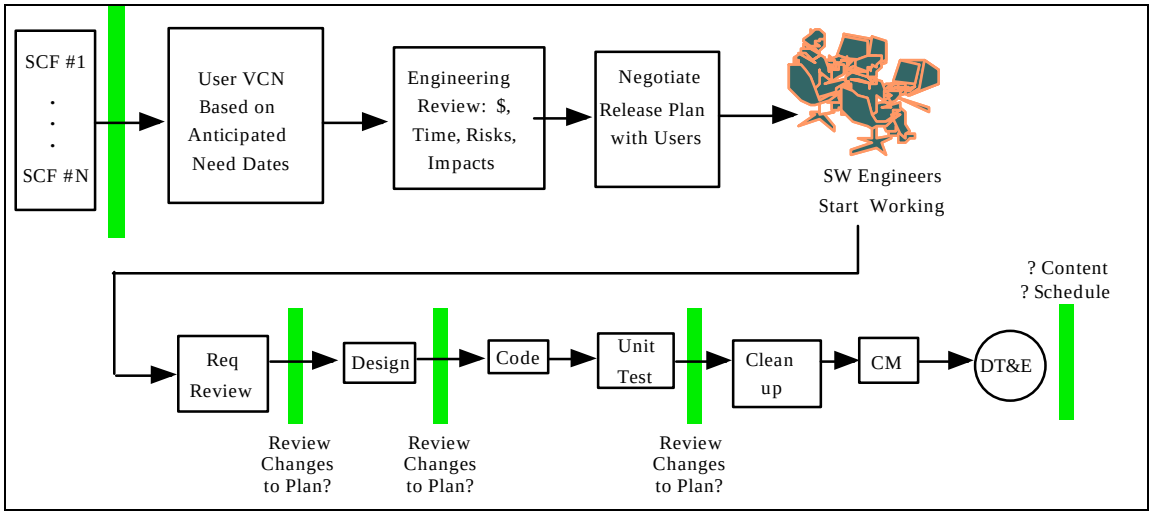


**Figure 3. Variable Staff/Variable Schedule Process**

## 3. Strategy Success Criteria and Data Collection

Basili and Rombach [16] developed the goal/question/metric paradigm to help project managers tailor their software process to organizational goals and their operational environment. Alexander [17] proposed a set of 20 criteria grouped into 5 categories (personnel, problem, product, resource, and organization) to

help a project manager select a process model. Whitten, Bentley, and Barlow [18] offer the PIECES criteria (Performance, Information, Economics, Control, Efficiency, and Service) for improving systems and processes. Thus, the criteria that should be used in making strategy comparisons is open to debate.

The criteria presented here are by no means definitive and are not unique to software maintenance process models. However, the particular criteria were selected with specific reference to these strategies using literature research and the informed judgments of managers in the field. In some cases, the application may result in the elimination of inferior strategies. However, the relative weight to be placed on different criteria depends on the context in which the strategy is being implemented (as described above). Thus, all of the models may survive the comparison process, each with its own area of applicability.

The criteria we chose are:

- Process Performance
- Economics
- Process Efficiency
- Product Quality

We collected data to calculate measures for each of the criteria between November 1994 and December 1996 on six different programs. Three programs were using the fixed staff/variable schedule strategy, two were using the variable staff/fixed schedule strategy, and one used the variable staff/variable schedule strategy.

Process performance has two components: (1) throughput and (2) "priority" cycle time. Throughput was measured as the average number of changes delivered to the user per year. The "priority" cycle time was measured as the 75th percentile of the number of days required to deliver a priority change. The cycle time was calculated as

$$Cycle\_Time = Change\_delivery\_date - \mathrm{Re}\,quirement\_validation\_date \qquad (1)$$

The 75th percentile was chosen because the cycle time distribution is skewed with the majority of the changes taking a short period of time, while a few take a long time to complete. Figure 4 is a cumulative distribution plot from one of the programs we observed. To read this plot, find 75% on the y-axis, (this is just to the right of the knee of the curve) read down to 135 days on the x-axis. This means that 75% of the time this strategy delivers high priority changes in less than 135 days from the time the requirement is validated. Note: The average for this distribution is 129 days with a standard deviation of 148 days.

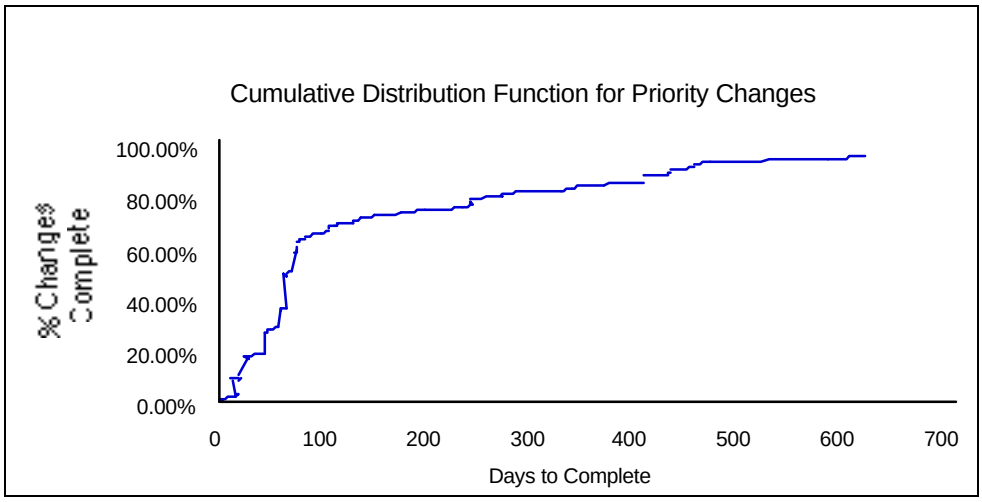


**Figure 4. Example Cumulative Distribution Function of Priority Change Request Times for One Maintenance Strategy**

For all three strategies the priority of the change was determined by the system user as either emergency (highest priority), urgent (high priority), or routine (low priority). If the user determined the change was either emergency or urgent priority, its cycle time contributed to the percentile reported.

The economics of the process is calculated as the average cost per change delivered for each process. The cost is tracked closely per release and includes all expenditures including engineering, management, travel, configuration management, hardware maintenance on the software development system, and financial overhead. The cost per change is calculated as

$$Cost\ \_\ per\ \_\ change\ = \frac{Total\ \_\ dollars\ \_\ spent}{Total\ \_\ changes\ \_\ delivered} \qquad (2)$$

For example, Figure 5 shows the total cost for each release completed during FY96 for one organization and the number of changes delivered in each release. The average cost per change is calculated by

cost per change = (13135 + 12417 + 12300) ÷ (197+183+108) = 77.56

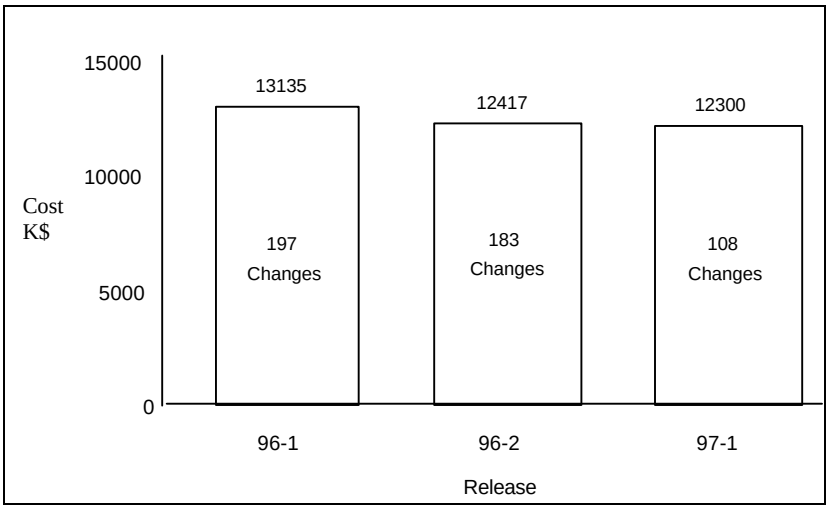


**Figure 5. Cost per Release for One Strategy for One Year**

The efficiency of the process is defined in terms of two attributes: (1) the percent of schedules met, and (2) the percent of releases that experienced requirements changes after the contents were defined. The percent of schedules met is calculated as:

$$Schedule\ \_\ \% = \left( \frac{\#\ deliveries\ \_\ accepted\ \_\ on\ \_\ or\ \_\ before\ \_\ planned\ \_\ date}{Total\ \_\ number\ \_\ of\ \_\ deliveries\ \_\ made} \right) * 100 \quad (3)$$

Naturally, the target for each strategy is 100% of the schedules met. Figure 6 shows the raw data for one strategy. In this plot 100% means that the product was delivered on the planned date (e.g., P5), a value less than 100% means the product was delivered early (e.g., C2) and a value greater than 100% means the product was delivered late (e.g., C1). For this strategy, only 24% (4 out of 17) of the planned delivery dates were met during the fiscal year plotted.

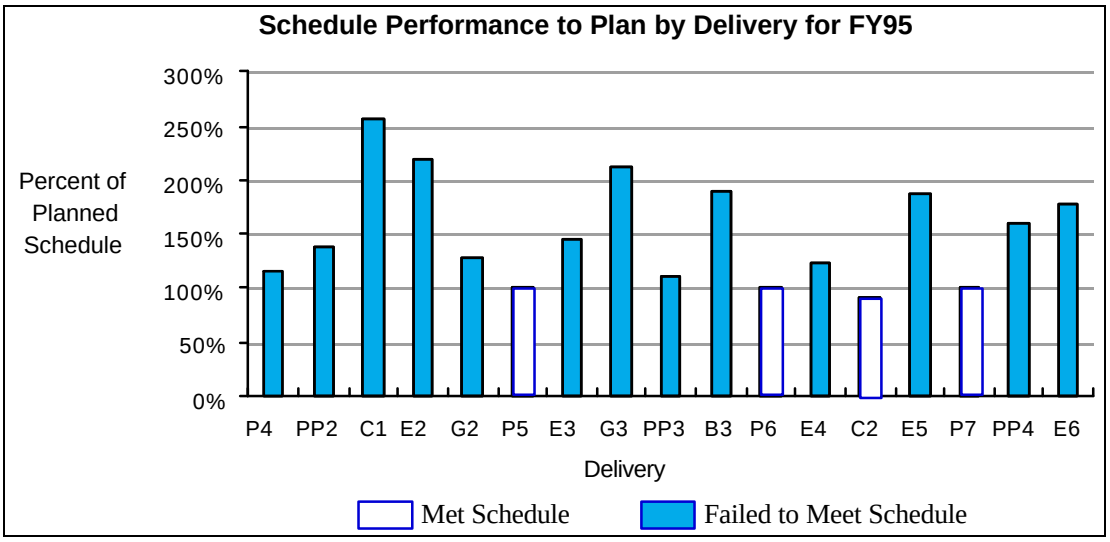


**Figure 6. Schedule Performance to Plan for One Strategy During FY95**

The second attribute of efficiency is the percent of releases that experienced no content changes after the release plan was established. This is calculated as

$$Delivery\_Efficiency = \left( \frac{\#\ deliveries\_without\_content\_changes}{Total\_number\_of\_deliveries\_made} \right) * 100 \qquad (4)$$

Naturally, the target for each process is 100%, but this is rare. In fact, Figure 7 shows that for this strategy 8 of 13 deliveries experienced content change. Ambiguity in the requirements sometimes causes changes in scope. Customers changing their minds about priorities as well as schedule and quality pressure often result in content additions or deletions from the plan.

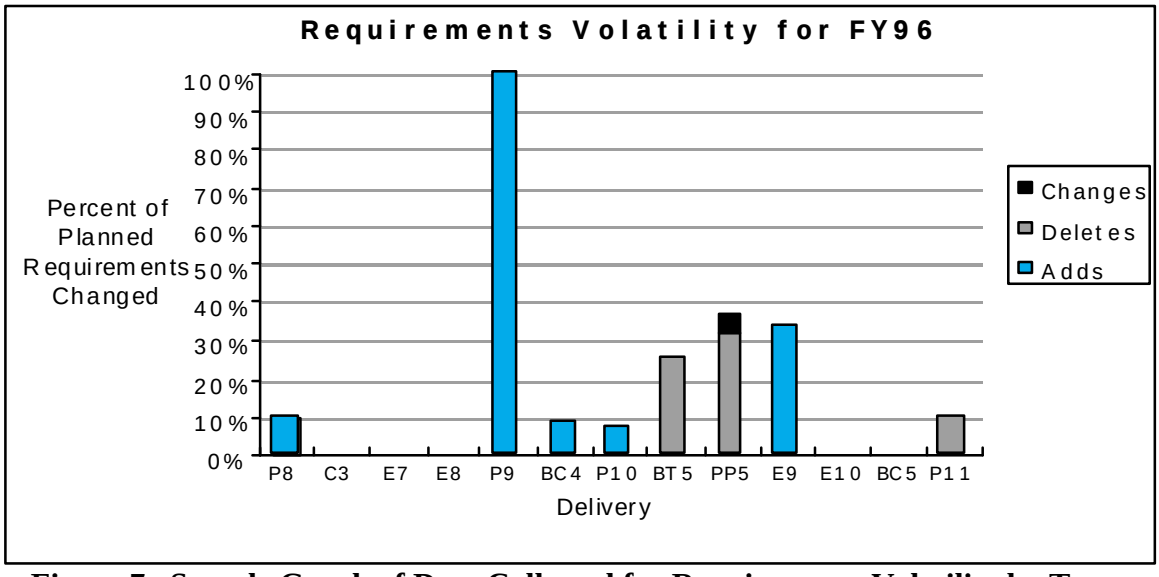


**Figure 7. Sample Graph of Data Collected for Requirements Volatility by Type**

The quality of the product produced by each strategy was measured as the percent of changes with customer found defects during operational acceptance testing. It is calculated as:

$$Quality = 1 - \left( \frac{\#\_changes\ \_with\ \_defects\ \_identifed\ \_during\ \_customer\ \_testing}{Total\ \_number\ \_of\ \_changes\ \_delivered} \right) * 100 \quad (5)$$

Again, the goal here is 100%. The goal of each process is to deliver the customer a software release that will execute the operational scenarios without failure. Customers often use different test scenarios than the developers and have different interpretations of failure, thus the metric goal was not met by any of the processes. Figure 8 shows the quality metric for four releases of one process. This is also a measure of the amount of rework required by the process since the defects found during customer testing are often written up as new change requests and must move through the process again.

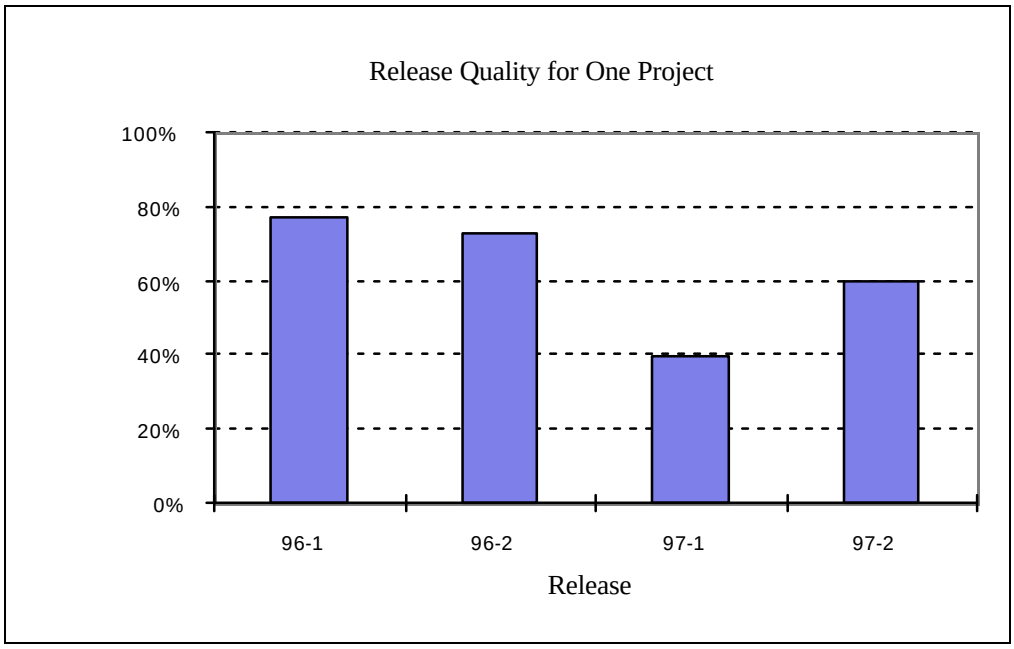


**Figure 8. Sample Release Quality Metric for One Project**

## 4. Results

Using the measurements defined in the previous section, the three maintenance strategies were compared after two years of operation. Table 1 displays the results. The table contains a row for each strategy, a row that describes the goal for each measure, and a column for each measurement collected. From the table, we observe the management tradeoffs that should be

considered in choosing a strategy. For example, a lower schedule achievement rate corresponds to higher quality in the end product.

**Table 1. Summary Process Comparison Table**

| | Process Performance | | | Efficiency | | |
|---|---|---|---|---|---|---|
| Criteria / Process | Average Throughput (Changes Delivered per year) | Priority Change Response Time (Days) | Economics (K$ per Change) | % Schedules Met | % Releases with no Content Change (total releases) | Quality {1- (number of defects per change)} |
| Goal | High | Low | Low | 100% | 100% | 100% |
| Fixed Staff / Variable Schedule (3 sites) | 190 | 111 | 78 | 100%* | 70% (10) | 60% |
| Variable Staff / Fixed Schedule (2 sites) | 445 | 246 | 104 | 72% | 22% (23) | 71% |
| Variable Staff / Variable Schedule (1 site) | 136 | 135 | 41 | 43% | 40% (30) | 94% |

*always 100%

Table 1 shows the variable staff/fixed schedule had a higher throughput than the other two strategies. The strategy was also able to absorb a large percentage of requirement's volatility and still meet 72% of their schedules. However, the variable staff/fixed schedule strategy had the highest cost per change and longest priority cycle time. This makes some sense since the schedule pressure and the requirement's volatility require more staff and time to complete. Since this process is schedule driven, the urgent changes are made “in the next available” planning cycle, rather than incorporated into the current release. Because releases are planned at least one year in advance the cycle time is at least 365 days for urgent changes. The emergency changes (which are incorporated into the current release or become their own release) bring the cycle time down.

Although the variable staff/variable schedule strategy delivers the fewest changes per year, it appears the most cost-effective and it delivered the highest quality products over the time period. The higher quality is attributed to the impact of the multiple reviews and the addition of a requirement review missing from the other two processes.

As expected, the fixed staff/variable schedule process delivered the best “on-schedule” performance, while the variable staff/variable schedule process met

only 43% of their plans. Interestingly, the fixed staff/variable schedule process showed 30% of the releases had a content change. This means that changes were added or deleted to the release during the "freeze" period on 3 of the 10 releases observed.

## 5. Conclusions and Observations

Three different software maintenance management strategies were described and compared on four attributes: performance, economics, efficiency, and quality. Six measures were used to define the four attributes. All three processes have attributes that make them appealing to software maintenance managers:

- The fixed staff/variable schedule strategy allows managers to prioritize changes and allocate resources based on individual engineering skills. It also supports the definition of delivery dates based on the sustaining organization's productivity.

- The variable staff/fixed schedule strategy handles requirement volatility and delivers a large number of changes per year.

- The variable staff/variable schedule strategy requires more formal planning and management oversight than the other two strategies, but experienced lower cost per change and highest customer satisfaction.

- Although we have observed these three strategies in operation over two years, our results must be caveated as follows:

    - Individual changes vary greatly in scope. For example, some changes are simple database updates while others require an 8-10 page specification and a thousand source lines of code to implement. This variability affects the economics of the strategy, the throughput, and the quality. We assume that because of the large number of total changes for each strategy that the statistical law of large numbers takes affect and that there was a similar distribution of change difficulty across the six programs.
    - User expectations and participation in the release process differs. For one of the organizations we observed, the customer rarely participated in the version release process. Others had user teams support the planning and execution of the release. This participation affects the change prioritization, the release volatility, and the customer satisfaction with the release. We believe that customer participation should be encouraged in all strategies.
    - The goals of the program management affect the outcome of the process. For example, if a program manager emphasizes flexibility, then the key

measures for the strategy would be change response time or throughput. Whereas, if a manager emphasizes economics, then cost per change becomes the key strategy measure.

## 6. References


1. IEEE, Standard for Software Maintenance, IEEE Std 1219, IEEE Computer Society, Los Alamitos, CA, 1993.
2. Lientz, B. P. and E. B. Swanson, "Characteristics of Application Software Maintenance," *Comm of ACM*, vol. 21, no. 6, Jun. 1978, pp. 466-471.
3. Parikh, G., The Guide to Software Maintenance, Winthrop Publishers, Cambridge, Mass, 1982.
4. Pigoski, T. M., Practical Software Maintenance, John Wiley & Sons, Inc., New York, NY, 1997, pp. 29-31.
5. Moad J., "Maintaining the Competitive Edge," *Datamation*, vol. 36, no. 4, Feb. 1990, pp. 61-66.
6. Card, D. N., D. V. Cotnoir, and C. E. Goorevich, "Managing SW Maintenance Cost and Quality," *Proc. Intl. Conf on SW Maint.,* Sept. 1987.
7. Chapin, N., "The Software Maintenance Life-Cycle," *Proc. Intl. Conf on SW Maint.,* 1988.
8. Hariza, M., J. F. Voidrot, E. Minor, L. Pofelski, and S. Blazy, "Software Maintenance: An Analysis of Industrial Needs and Constraints," *Proc. Intl. Conf on SW Maint.,* Orlando, Fl., 1992.
9. Software Engineering Institute (SEI), "Software Process Maturity Questionnaire Capability Maturity Model, version 1.1," Carnegie Mellon Univ., Pittsburgh, PA, 1994.
10. Boehm, B., Dec. 1976, "Software Engineering," IEEE Transactions on Computers, Vol. 12, No. 25, pp. 1226-1242.
11. Martin, J. and C. McClure, 1983, Software Maintenance: The Problem and Its Solutions, Prentice-Hall, Englewood Cliffs, NJ.
12. Sharpley, W. K., Nov. 1977, "Software Maintenance Planning for Embedded Computer Systems," proc. IEEE COMPSAC, pp. 520-526.
13. Parikh, G., 1982, "Some Tips, Techniques, and Guidelines for Program and System Maintenance," Techniques of Program and System Maintenance, Winthrop Publishers, Cambridge, MA, pp. 65-70.
14. Basili, V. R., L. C. Briand, S. Condon, Y. Kim, W. L. Melo, and J. D. Valett, March 1996, "Understanding and Predicting the Process of Software Maintenance Releases," proc. 18th Intl. Conf. on SW Eng., Berlin, Germany.

15. Alexander, L. C., and A. M. Davis, Sept. 1991, “Criteria for Selecting Software Process Models,” proc. 15th Annual Intl. Computer Software & Applications Conf. - COMPSAC 91, Tokyo, Japan, pp. 521-528.

16. Basili, V. R., and H. D. Rombach, 1987, “Tailoring the Software Process to Project Goals and Environments,” proc. 9th Intl. Conf on Software Engineering, Washington, DC, IEEE Computer Society Press.

17. Alexander, L. C., 1990, Selection Criteria for Alternate Life Cycle Process Models, Software Systems Engineering M. S. Thesis, George Mason University, Fairfax, Va.

18. Whitten, J. L., L. D. Bentley, and V. M. Barlow, 1994, Systems Analysis and Design Methods (3rd ed), pp. 99, Irwin Publishing, Englewood Cliffs, NJ.