SOFTWARE PROGRESS TRACKING SYSTEM

T. Michael Moriarity
AAI Corporation
Cockeysville, Maryland

ABSTRACT

The increasing complexity of software systems combined with the requirements for structured and modular designs have increased many fold the number of software elements developed and delivered on recent simulator programs. The increased number of elements plus the traditionally "soft" milestones used to measure progress has made monitoring software development and predicting future progress time consuming, subjective, and often unreliable.

A software progress tracking system which uses an earned point scheme has been successfully used to monitor software development on several large simulator programs. Points are assigned for each step in the software development cycle on a per element basis. The steps are "hard" milestones in which a generated product is accepted by program management. As the products are accepted the associated points are earned. The ratio of earned points to total possible points is compiled on an element, functional area, or total software system basis to determine progress achieved. A report generator program, usually resident on the simulator computational system, tabulates the data in a variety of management reports.

The system as implemented is flexible, highly automated, and is closely coupled to configuration management systems and software quality assurance procedures to ensure validity of data. The accumulated point values are quickly ascertained, objective, and based on the current state of program development. Simple calculations or comparisons of the accumulated point values provide an accurate measure of progress, deviation from schedule, and prediction of future progress.

INTRODUCTION

A large body of literature addresses the functions of planning, organizing and directing software development projects. Many texts and articles comprehensively discuss the initial stages of a project. The need for detailed requirement specifications, development plans and development specifications are well documented. The importance of clear organizational structure and team responsibilities are well supported. The techniques of top down, stepwise refinement of design and the requirement to review and validate designs early in a program are well known. Conscientious application of these management tools help ensure that the completed software product will meet its performance requirements in a timely manner.

These tools and techniques emphasize functions performed early in the life of a project. Less information is available on the on-going management function of control. Control can be thought of as a three step process: an attribute or characteristic of interest is measured, the measured value is compared with an expected or baseline value, and an appropriate action is taken if an unacceptable deviation exists. Any number of items of interest during software development may be controlled in this manner. Development time, development costs, computer memory usage and computer time are some of the more common items.

The purpose of this paper is to describe a method of measuring performance of the software development team and comparing the measured performance to a baseline schedule. Performance here refers to the effectiveness of software development personnel in meeting established schedule and cost targets. By providing an objective, timely measure of actual performance with a comparison to expected performance, project management will have the means to pinpoint schedule and cost deviations thus enabling them to take action to assure schedule and cost targets are met.

A performance measurement scheme should meet several criteria. First and most importantly, the scheme should be objective. The person claiming performance should not be required to estimate degree of completion. Likewise, the person monitoring performance should know exactly what a performance measurement represents. Ideally, the state of development should be sufficiently visible and the measurement means sufficiently clear to enable any project member to make the actual measurement.

Secondly, the scheme should measure performance in accomplishing the real task, i.e., the development of deliverable software. Further, the resolution of the measuring scheme should be sufficiently fine to measure incremental progress on a weekly or monthly basis and the measurement should be timely in that it measures the current state of development. Providing accurate, current performance information on a periodic basis can be a positive motivating factor for a programming staff.

Lastly, the scheme needs to be efficient. It should require minimal resources to collect, collate and report performance data and should require minimum time to interpret the results. Systems which require constant inputs from the programming staff, updates by clerical personnel, or integration of large amounts of data by management are not used.

TYPICAL METHODS OF MEASURING PERFORMANCE

Performance in software development is measured typically either by estimating percent completed of a task or by counting the number of predetermined milestones which have been reached. In either method a schedule of tasks and/or milestones is used as a baseline with which measured performance is compared.

In the estimate of percent completed method the person actually doing the work estimates the percent of the work which has been accomplished in reaching a milestone or completing a task. The percent completed method has several faults. The major fault is that the measurement is subjective. The manager is asking a person with a vested interest in completing the task as early as possible to make an educated guess as to how nearly complete he is. Most people tend to be optimistic in their ability to complete a task – particularly if their manager subtly encourages optimism. The old bromide of a task being 95% complete for months is all too true.

While not necessarily a characteristic of the percent completed method, a potential shortcoming of this method when used with tasks rather than milestones, is the definition of completion is not always stated. Therefore, the person making the estimate may have one perception of what the task includes, while his manager may have another. Hence when the programmer states the task is 100% complete -- written, tested and documented – the manager may have an unpleasant surprise when he asks to see the Installation Guide. Therefore, since the end of the task may not be clearly defined, the estimates of completion may be quite inaccurate.

Since the estimates are subjective, the interpretation of the results may also be subjective. In trying to ascertain the degree of completeness of a job, a manager may ask who made the estimate and then apply a "correction factor" to the estimate for that person to get a number he feels comfortable with.

The second method, or milestone method, attempts to alleviate these problems by defining specific milestones which must be met and measuring performance by summing the number of milestones which have been met. This method is much more objective, tends to describe the overall task more fully and as a result is easier to interpret. The shortcomings of this method are more in the area of resolution of the measurement versus the efficiency of collecting, collating and presenting the results in a meaningful way.

In order to get the resolution of measurement fine enough to show incremental progress on a periodic basis, a large number of milestones need to be defined. However, the large number of milestones makes it more difficult to collect and present the data in a timely and meaningful way. A common method is to present the data on bar graphs, but on a large project with thousands of milestones, the upkeep of bar graphs can be a slow, expensive effort.

Another potential problem is that the milestone may not accurately reflect the real task. If care is not taken to define the milestone, the milestone may not be based on deliverable items but based on something which appears to show progress such as lines of code generated. Also, if the milestones are not carefully chosen, it may be difficult to determine if the milestone has been reached.

POINT SYSTEM

A method of overcoming these problems is a point system, which has been successfully used on several large simulator programs. The point system is really an extension to the milestone system which lends itself to automation. In its simplest form it is assumed that each software module goes through a similar development process and there are a number of clearly identifiable milestones within that process. For the purpose of illustration, assume ten modules will be developed and four milestones will define the development process. The milestones may represent design reviewed and accepted, code walkthrough complete, test results verified, and module released.

In the simple case each milestone for each software item is worth a point. In the case of the system with ten modules forty (40) points can be earned. As part of each design review, code walkthrough, test verification or release audit, the milestone is achieved and the corresponding point earned. By listing all of the modules and milestones achieved (points earned) in a computer file, and creating a few simple report generators, an objective, accurate, and timely measure of performance can be acquired. Figure 1 shows what a simple status report might look like.

SOFTWARE STATUS REPORT

	DESIGN	CODE	TEST	RELEASE	POINTS EARNED
Module A	1	1			2
Module B	1				1
Module C	1				1
Module D	1	1	1		3
Module E	1	1			2
Module F	1				1
Module G	1	1			2
Module H	1	1	1	1	4
Module I	1				1
Module J	1	1			2
TOTALS	10	6	2	1	19

PERCENT COMPLETE = 19/40 = 48%

Figure 1. Simple Status Report

This simplified scheme works well for a homogeneous set of modules where all modules are of the same complexity and each of the milestones represent an approximately equal amount of work. Through an introduction of weighting factors, modules of varying complexity or milestones representing unequal effort to complete can be easily handled.

Before this and other extensions are discussed, however, a brief description of implementation is in order. The heart of the system is a computer data file and a few simple report generators. The data file is simply a collection of records, one for each item which is to be tracked, which contains fields to indicate whether a particular milestone has been met or not. Usually, it is advantageous to include fields that allow for description of the item, responsible analyst, work package identification and various file identification fields. Figure 2 shows a sample record layout. Often such a file will serve multiple uses particularly if a few additional fields are added. Typical uses are Family Tree Definition, Specification Cross References, Configuration Control List, Documentation Cross Reference, and any one of a number of uses where a comprehensive list of deliverable software items are needed.

Layout

FILE NAME	SR NAME	RA	WP NUMBER	TREE DESIG	DCTR
DESCRIPTION					

File Name = Name of File as it exists on disk
SR Name = Name of Subroutine as it is called
RA = Responsible analysts initials
WP Number = Work Package Number
Tree Desig = Software Family Designation
DCTR = Status for Design, Code, Test and Release Milestones

Sample

FILE NAME	SR NAME	RA	WP NUMBER	TREE DESIG	DCTR
F.UDHEAD	UHEAD	MKM	17A	113206	11
DESCRIPTION					
PRINT HEADING FOR LISTING					

Figure 2. File Layout

Maintenance or updating of the file can be as straight forward as modifying records with a line editor or as complex as building a special purpose, interactive update program. Some means of limited access should be used to restrict unauthorized modification of the file, particularly if some of the other uses of the file are sensitive to change.

Once the file is updated to include an entry of the module under development, the milestone status fields are updated as the milestones are met. In some cases this may be a manual process, whereby once an event has occurred and the milestone achieved, a program librarian or other authorized person updates the status file. In other instances, in a more sophisticated system, a computer program could determine that milestone event has occurred (error free compilation or successful test run) and automatically update the milestone status.

After the file has been built, report generator programs are written to print the status.

For smaller projects, a program which simply prints each record, sums the earned and points defined, and calculates the percent points earned, may be sufficient. Larger projects may need several reports for different subsystems or summary reports which emphasize change.

EXTENSIONS

A number of extensions can be added to the scheme as described so far. The first is to add a method of weighting modules and/or milestones. While weighting all modules equally on a large program, where many (over 1000) modules exist appears to give good results, smaller programs with few modules may need to weight the modules to give a sufficiently accurate measurement of performance. Also, depending on the level of visibility of the measuring system and the attitude of the personnel involved, there may be a tendency to do all the "easy" modules first to show early performance.

A similar argument can be made for weighting milestones. Depending on the acceptance criteria to meet a milestone, some milestones may involve more work than others, therefore achieving those milestones represents accomplishing a greater amount of work than in meeting other milestones. Further, there may be instances where a combination of module weight and milestone weight may interact. An example is a module which was previously written on another project in a different language. The amount of design work for that module may be considerably less than a module designed from scratch, but amount of effort to code the routine might be more since an unfamiliar language may be involved.

The weighting scheme is easily implemented by assigning points to each milestone for all modules. Then as a milestone is earned, the assigned points are added to the totaled earned and divided by the total defined points to compute percent completion. The number of points assigned to each milestone is in proportion to the difficulty in achieving the milestone, and, in fact, relates directly to the estimated number of hours needed to complete the milestone. In assigning points it is recommended that points first be assigned to each of the modules and then reapportioned to the milestones.

A second extension is to add selecting and sorting options to the report generator programs. Selecting options allow the user to select all entries in the file by some field such as work package number, file name, software family tree component, or responsible analyst. Once the entries of interest are selected, the sort option allows the user to order the entries by some key. The points earned and points defined are summed from the selected entries and the percent complete calculated. Therefore, reports can be printed listing all modules and percent complete for a certain analyst, work package, or other selected criteria. It has been found valuable to allow boolean operations on selection fields (Analyst A AND Subsystem B) and to provide for major and minor sort fields (List modules in alphabetic order by analyst).

A third extension which has been useful is to add target dates and actual completion dates to each module record. In this extension the individual milestone status fields are replaced by two dates. The first date field is a target date as to when the milestone should be met. The target dates do not have to be used for all modules or milestones but are useful where an interdependency exists between a particular module milestone and some other element in the system. These interdependencies may exist in the design stage to some extent, but they become very important during the integration phase of a project.

The actual completion date field becomes a flag as to when the milestone is achieved. By adding up the points assigned to a milestone that have an actual date entered in the file the percent complete can be computed.

Using the two date fields has two advantages: schedule interdependence can be monitored and a historical record exists for future analysis. By making the date fields selectable and sortable, additional interesting reports can be generated. Assuming that an integration milestone has been identified, a list of all modules of interest can be selected by WBS work package number, family tree identification or individual module name. Target dates for the milestone of interest can then be entered. As the date of the integration milestone comes closer, lists of all modules of interest which have a particular due date and have not been completed can be provided to the responsible analyst or work package manager. Judicious use of these lists on a periodic basis can be used to monitor and motivate the programming staff to assure the milestone is met. Usually, several of these lists in various stages are active at once as key milestones are coming up. It has been found that choosing approximately one major milestone a month and starting the list several months in advance of the target date is very effective. More milestones than this tends to set up multiple or conflicting goals for the individual analysts. Also the lists need to be started well enough in advance to allow suitable time for the work to be completed and institute work-arounds if problems arise.

It should be noted that the meeting of these interdependency dates is really separate from performance measurement. It is possible that in a given subsystem the performance may be fully adequate, say 75% complete, but a key integration event may have been missed. The manager must be aware of both elements. If performance is where it should be, but an integration event has been missed, it may mean his people are not concentrating on the right items and need to be re-directed.

ROLLING BASELINE

A potential problem with the point system described thus far has to do with an effect known as a rolling baseline. The rolling baseline occurs over the life of a program as new items are continually defined and added to the status file. This has the effect of changing the baseline which causes percent complete to vary independently of milestones earned. During periods when few new items are added to the file, the percent complete accurately reflects real performance. At other times, as new items are added as quickly as previously defined milestones are met, reported progress tends to flatten out. In some instances where more new items were added than old items completed, negative progress is reported.

This problem is overcome by freezing the baseline for a unit of work or work package basis and reporting progress on the unit. That is, once a package of work is defined, no new points are allocated to the package. If for some reason it is decided certain modules have to be split up for sake of modularity or computing efficiency, *the points are likewise split up in a replanning* effort. In the instances where the scope of work changes due to an oversight or contract change, the effort is reprogrammed and either new work packages are created or existing work packages are expanded with a corresponding increase of points.

This has the effect of measuring performance on active or open work packages only and not on the system as a whole. However, since performance is being compared to an established schedule which is also made up of units of work, the comparison is valid and useful.

REPORTS

Several informative detail reports and summary reports can be generated from the data file. The most encompassing detail report, of course, is a listing of all elements. Such a list may be useful in creating inventory lists of software items to be delivered and might be used during Physical Configuration Audits. Other lists may be sorted and/or selected by work package or family tree identification number. Such lists show status of specific modules within subsets of the Work Breakdown Structure or functional items of the system. Other sorts or selections by a responsible analyst shows status of a particular individual's effort. Figures 3 and 4 show a sample detail reports.

The *summary reports* function as its name indicates. They sum up the information generated by the detail reports and simply total the items in each selected category. For example, a detail status listing of all elements within a work package can generate a summary that indicates the total number of elements and the number of elements that have met each milestone. Each of the milestones can also be expressed as a percent. The summary reports can then be used to report performance by whatever category is needed, i.e., work package, family tree element, responsible analyst, event milestone, etc. Figures 5 and 6 show sample summary reports.

Collecting data from several summary runs allow rates of completion to be calculated upon which trends or predictions of future performance to be made.

INTERDEPENDENCY STATUS REPORT

FILENAME	ID	RA	CLASS	DESCRIPTION	DESIGN	CODE	TEST	RELEASE
F.UDHEAD	DF-U150	MKM	U	PRINT HEADING FOR DELTA LISTING (CONFIG)	--/--/-- 01/27/83	--/--/-- 02/08/83	--/--/-- 03/15/83	04/15/83 04/21/83
F.UDLIST	DF-U151	MKM	U	PRINT DELTA LISTING (CONFIG)	--/--/-- 01/31/83	--/--/-- 02/10/83	--/--/-- 03/15/83	04/15/83 04/21/83
F.UDLTST	DF-U152	MKM	U	START UDELTA SUBTASKING (CONFIG)	--/--/-- 01/31/83	--/--/-- 02/15/83	--/--/--	04/15/83
F.UDMAT	DF-U153	MKM	U	CHECK BUFFERS FOR MATCH (CONFIG)	--/--/-- 01/14/83	--/--/--	--/--/--	04/15/83
F.UDMOVE	DF-U154	MKM	U	MOVE DATA INTO MEMORY (CONFIG)	--/--/-- 02/02/83	--/--/-- 03/01/83	--/--/-- 04/04/83	04/15/83 04/11/83
F.UDOPT	DF-U155	MKM	U	SET OPTIONS IN DELTA (CONFIG)	--/--/-- 02/01/83	--/--/-- 02/28/83	--/--/-- 04/14/83	04/15/83 04/11/83

Figure 3. Detail Interdependency Listing

WORKPACKAGE STATUS REPORT

WP: TACTICS LIBRARY SOFTWARE

MANAGER: NFB

			MILESTONES				MODULE STATUS		
WORK PACKAGE	FILENAME	WEIGHT	DESIGN	CODE	TEST	RELEASE	STATUS CODE	SCORE	% COMPLETE
173F	F.LEDCPY	8	2	2	2	2	3	4	50
173F	F.LEDEL	8	2	2	2	2	3	4	50
173F	F.LEDFIL	44	11	11	11	11	1	11	25
173F	F.LEDINF	20	5	5	5	5	1	5	25
173F	F.LEDPRT	12	3	3	3	3	7	12	75
173F	F.LIBEDT	16	4	4	4	4	3	8	75
173F	F.LIBGEN	28	7	7	7	7	15	28	100
173F	F.LTACGN	16	4	4	4	4	3	8	50
173F	F.LTACID	8	2	2	2	2	15	8	100
173F	F.LTASTA	32	16	0	0	16	7	16	50
173F	F.LTCMPR	16	8	0	0	8	15	16	100
173F	F.LTCMST	56	28	14	14	0	0	0	0
173F	F.LTCVRT	12	3	0	0	3	0	0	0
173F	F.LTGNUM	12	3	3	3	3	0	0	0
173F	F.LTINIT	12	3	3	3	3	0	0	0
173F	F.LTMDID	16	4	4	4	4	0	0	0
173F	F.LTREC	32	8	8	8	8	0	0	0
173F	F.LTSSTM	48	24	6	12	6	1	24	50
173F	F.LTUCHK	8	4	1	2	1	3	5	63
173F	F.LTUCVT	12	6	2	3	1	7	11	92
173F	F.LTVALU	8	4	1	2	1	15	8	100
TOTALS:	21	424	106	106	106	106		168	40

Figure 4. Detail Status Listing

STATUS SUMMARY

WORK PACKAGE: 1234

	DESIGN		CODE		TEST		RELEASE		TOTAL	
TOTAL ITEMS	24		24		24		24		96	
TARGET COMPLETE	10	42%	7	29%	3	13%	0	0%	20	21%
ACTUAL COMPLETE	9	38%	5	21%	1	4%	0	0%	15	16%
LATE	1	4%	2	8%	2	8%	0	0%	5	5%
LESS THAN 1 WEEK LATE		0		1		0		0		
1-2 WEEKS LATE		1		0		2		0		
2-4 WEEKS LATE		0		1		0		0		
4-8 WEEKS LATE		0		0		0		0		
MORE THAN 8 WEEKS LATE		0		0		0		0		

Figure 5. Summary Report

WORK PACKAGE SUMMARY REPORT

				MILESTONES				WP STATUS	
WORK PACKAGE	DESCRIPTION	MGR	WEIGHT	DESIGN	CODE	TEST	RELEASE	SCORE	% COMPLETE
173G	SCAN LIBRARY SOFTWARE	NFB	480	120	120	120	120	150	31
173H	PPG LIBRARY SOFTWARE	NFB	296	74	74	74	74	74	25
173K	EMITTER SCRIPTING: EMTR 1-50	NFB	2500	2250	250	0	0	1055	42
17A1	TD REPORTING CPPS	TJR	310	155	155	0	310	310	100
17A3	TD REPORTING SW DEVELOPMENT	TJR	1230	375	375	240	240	575	47
17A4	SCAN PROCESSOR DOCUMENTATION	TJR	1078	863	215	0	0	0	0
17A5	TIMS, DEBUG, SVL DOCUMENTATION	TJR	7420	6550	870	0	0	3465	47
17A7	SOFTWARE DEV TOOLS DOCUMENT	TJR	4818	3563	1255	0	0	3563	73
			18132	13950	3314	434	434	9192	51

Figure 6. Summary Status Report

SUMMARY

The point system for performance measurement during software development provides an objective, accurate, efficient means of collecting and reporting performance data in an engineering field which often lacks visibility. The method uses data which is based on deliverable software items and which is collected as a *normal part of the development process.* The results are easily interpreted and can be presented in a number of formats and subdivisions. The scheme is flexible and can be modified to meet the needs of projects both large and small.

ABOUT THE AUTHOR

MR. MICHAEL MORIARITY is a Senior Design Analyst in the Electronic Warfare Operation at AAI Corporation. He currently is the Software Manager on the EF-111A Operational Flight Trainer Program. Mr. Moriarity holds a Bachelor of Science degree in Mathematics from the University of Minnesota and a Masters of Engineering Administration from George Washington University. Previous to his current position he was responsible for software development on the Navy Electronic Warfare Trainer System (NEWTS) and the defensive instructional subsystem on the B-52 *Weapons Systems Trainer. He has also had extensive* experience in developing software for automatic test systems.