Dynamic Software Life Cycle Model

Henry Neimeier
The MITRE Corporation
7525 Colshire Drive
Mc Lean, Virginia, 22102,USA

Abstract

This software life cycle model encompasses initial development, software upgrades, and error maintenance. The dynamic S**4 model is used to evaluate several different development and maintenance strategies. The impact of Integrated Computer Assisted Software Engineering (ICASE) tools on development and maintenance cost, schedule, and error rate is quantitatively evaluated. Alternative techniques for grouping errors and functions into releases are evaluated.

Dynamic Software Life Cycle Model

Purpose

The U.S. Department of Defense is faced with an ever increasing demand for software intensive systems on its combat platforms and supporting infrastructure. The combat effectiveness of these planes, ships, and tanks is highly dependent on the functionality and reliability of their embedded software. Given the huge investment in platforms and their operation costs, a small change in mission effectiveness has a major value impact. A recent General Accounting Office study reported: "Department of Defense (DOD) software costs total over $30 billion a year (estimated to be $42 billion by 1995), of which about two-thirds is for maintaining, upgrading, and modifying operational systems already in production. Today's major defense systems depend largely on the quality of this complex and increasingly costly software. In fact many major weapons systems cannot operate if the software fails to function as required."

Current commercial software parametric models provide a static cost estimate. This fails to address such dynamic elements as requirements change, greater communications overhead with increased staff size, staff learning, integrated testing and rework processes, and changing progress forecasts. Since decisions made in the software development phase have major impact on later maintenance and error fix phases, a time based approach to costing is warranted. Another factor impacting cost is the use of Integrated Computer Assisted Software Engineering (CASE) tools and structured techniques which can reduce software errors and simplify later functionality upgrades. However, the additional tool and tool training costs in software development must be weighed against productivity gains and later maintenance benefits. Oftentimes the software development organization is different from the maintenance organization. This can lead to less rigorous testing in order to meet contractual obligations of cost and delivery date. The result is that the maintenance organization inherits a far greater cost of fixing errors later in the life cycle. This paper documents the start of a research program to develop a new software engineering life cycle paradigm that spans the entire software life cycle.

Related work

Tarek Abdel-Hamid and Stuart Madnick developed a detailed system dynamics model of the software development cycle. Their book provides key parametric and relationship data and references to supporting research. The book notes that real progress in a software process is not generally known until the project is 80% complete and module and integration testing is performed. Then the number of errors can be assessed and rework scheduled. Personnel added later in the process can actually make the project later. This is due to the training burden put on the experienced staff which takes time away from development activities. In addition, a larger staff has a higher communication overhead. This can lead the software process to exhibit hysteresis. The initial project size estimate impacts staffing, communication overhead, and training requirements. A different initial estimate results in a different schedule and cost.

Nghia Nguyen extended the development model to a software training flight simulator. This was implemented in "Microworlds Creator" a subset of the "S**4" software package used in our research. This flight simulator game was used to train several managers in software development project management. Our project has similar goals for the entire software life cycle.

Kenneth Cooper showed the prime cause for late software delivery is the late discovery of software errors. In his work he found that the average module quality (% of configuration control modules not withdrawn for rework) for several U.S. commercial software projects was

68%. For DOD projects the quality level was 34%, which resulted in many rework cycles. Most of the excessive DOD schedule overruns can be explained by this quality factor.

Capers Jones provides an extensive data base of software development relationships. Included are the effect of staff experience on productivity and error rate, the effect of quality assurance personnel allocation on software error rates, and the effect of CASE tools and structured methods on productivity and error rates. Software size is measured in terms of function points in this book. This measure is largely language independent, and thus was chosen for our model. The data provided in Capers Jones work are averages over a large diverse set of software organizations. We used these averages as our default values for the initial model runs. Later the model will be tailored directly to a user's organizational data.

Environment and measures of effectiveness

A law of diminishing returns operates as successive errors are repaired. As errors are fixed, the error intensity decreases and the time between true error reports increases. False error reports are a function of platform usage and do not reduce with time. Thus through time, a larger proportion of effort is spent diagnosing these false error reports. With continual budget reductions, sufficient funds are not available to maintain all platform software. So a key question is when to remove a platform from software maintenance in order to free-up funds to maintain other platforms. This led to the need for assessing the dollar value of maintenance operations in our life cycle model. A key need of DOD software maintenance organizations is to justify their budgets in terms of war-fighting mission effectiveness. If the useful remaining platform life can be estimated, a dollar per year value can be calculated. Next the effect of software functionality and reliability on platform effectiveness is assessed. Some platforms have a greater software dependence than others. Most of these estimates are uncertain, so an analytic uncertainty modeling approach will be applied to encode the input parameter uncertainties and calculate the uncertainty in final output measures of effectiveness in the next model release.

Error repairs and functionality additions have a mission effectiveness value over the remaining platform life. Thus, early fixes and enhancements have the potential for a greater cost effectiveness impact. The summary measure of effectiveness in our model is the benefit-cost ratio. Cumulative benefits beyond that of the initial software development are divided by the cumulative error fix and software upgrade cost. Given the high platform investment and operations costs, benefit-cost ratios of 300 are common.

The initial single platform model was developed in "i think" and uploaded to S**4 for the game interface. Later releases will consider multiple platforms competing for shared maintenance center resources. In addition dynamic uncertainty analysis will be used. Both of these options are greatly simplified by S**4's array capabilities.

Development and upgrade process

Figure 1 is an extract of the development and upgrade sector from the model. Figure 2 presents development and upgrade sector results of a sample run. An initial "ReqtFP" requirement function point size estimate is put into the "FP" function point tasking level at start of initial development and for each scheduled upgrade. The rate at which function points are completed (ProdR) is determined by staff assigned and the labor productivity. Priority is given to processing initial function points over processing discovered rework. The quality level (Qcor) determines whether function points are completed (FPCompleted) or enter the rework (FPRework) cycle. Changes in requirements cause function points to flow back from "FPCompleted" and "FPRework" to "FP". The higher the proportion of effort allocated to quality assurance

(QualityAssurePr), the earlier that the rework is discovered (DiscTm), and moved to the discovered rework level (FPDiscRew). Depending upon the quality level (Qcor), the discovered rework either joins FPcompleted or "FPRework". The later occurs when errors are made in rework fixes. All function points are added to the total system function point count when a completion release threshold is met, or the next scheduled upgrade starts. The threshold is required because it takes an infinite time to complete all rework. At deployment time, the associated errors are added to the undiscovered errors level (Errors) of the error fix process. Figure 2 shows a sample time history of "FP", "FPCompleted", "FPRework", and "FPDiscRew" levels. For the sample values used, there are times between upgrades when the sector is idle and all personnel are assigned to the error fix process (times when figure 2 function points are zero).

The functionality, estimated by the number of function points, deployed in aircraft software has grown significantly in the past ten years (15% to 35% per year). This information is captured with the potential function points level. If an aircraft's software is frozen, platform mission effectiveness decreases relative to other aircraft that continue enhancing software. Figure 3 shows the comparison of system function points deployed versus potential function points. The "FPCompleted" level shows the next upgrade in process. The mission effectiveness value increases when an upgrade is deployed (SystemFP).

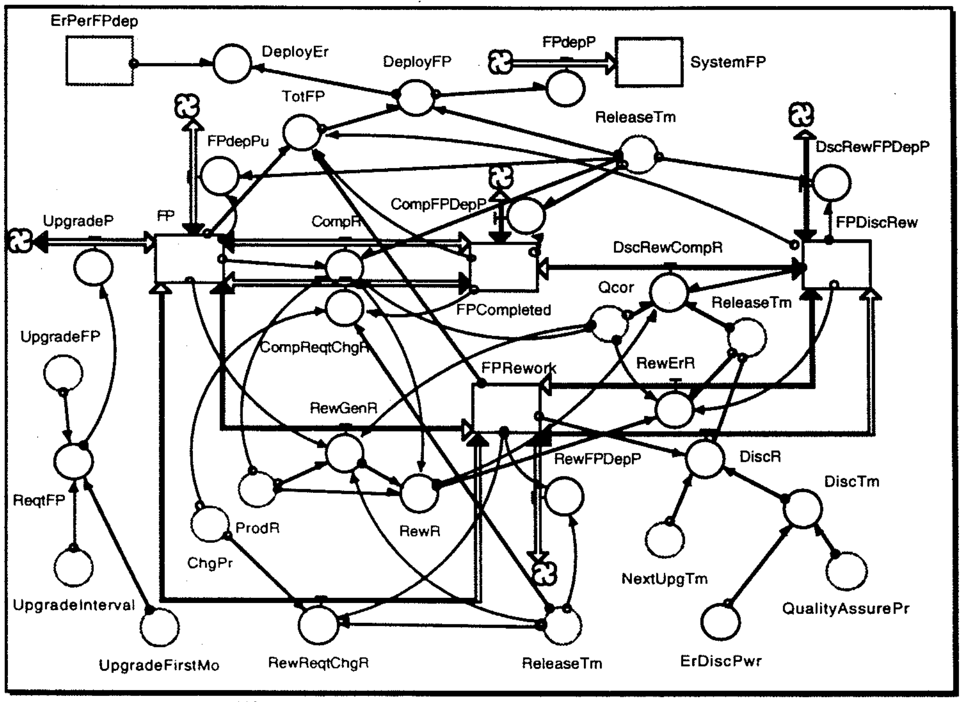


Figure 1. Development and Upgrade Sector

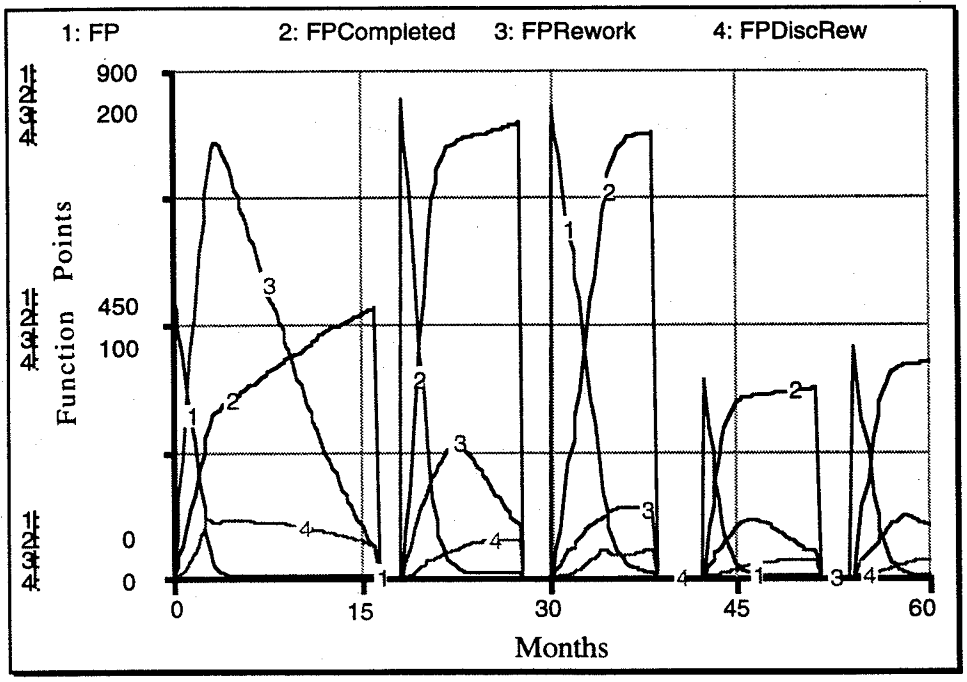


Figure 2. Sample Development and Upgrade Process

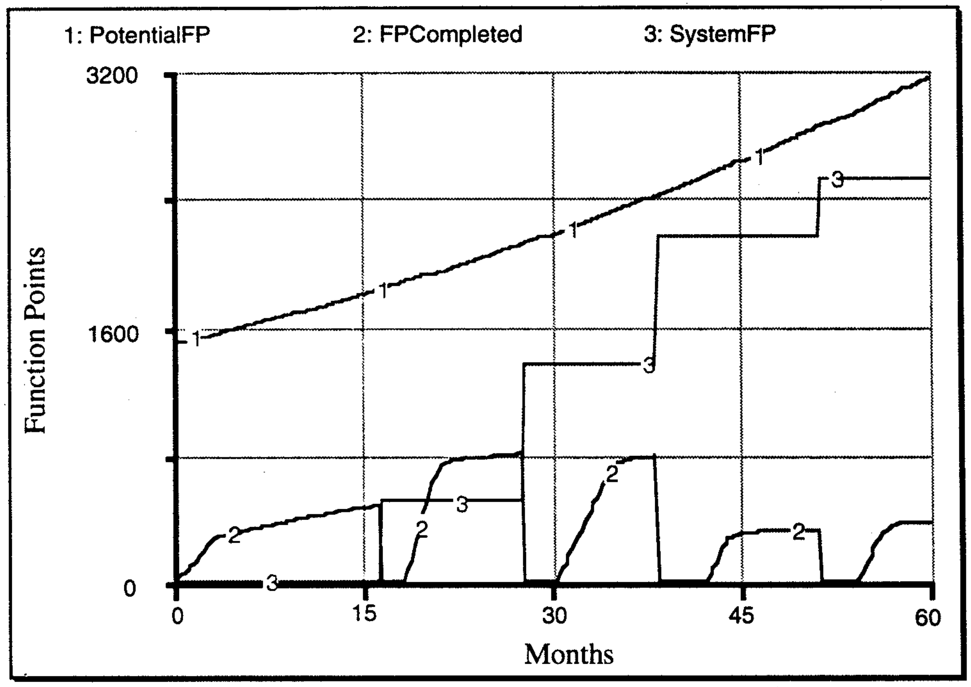


Figure 3. Sample Functionality

Error fix process

The error fix process sector is shown in figure 4 and a sample run is shown in figure 5. When a software release is deployed, the associated undiscovered errors are added to the "Errors" level. Errors are discovered and join the "Diagnose" level based on platform usage and the average error discovery months. In the diagnose process, errors are duplicated, severity assessed, fix resources evaluated, and results reported to the approval process. The approval process sorts errors into critical (immediately fixed), upgrade (fixed in next upgrade), or ignore (not worth the effort). Critical errors are fixed (CritErFix) tested (CritFixTest) and deployed (CritFixDepR) as soon as possible. Some of these fixes fail testing (CritTestFailR) and rejoin the "CritErFix" level. The process of fixing both critical and upgrade errors can insert new errors in the software. These new errors (FixErR) join the undiscovered errors (Errors) level. Figure 5 shows a sample run. Note that the lag between successive error fix process levels is a function of staff assigned and labor productivity. Labor productivity is a function of staff experience and tools employed.

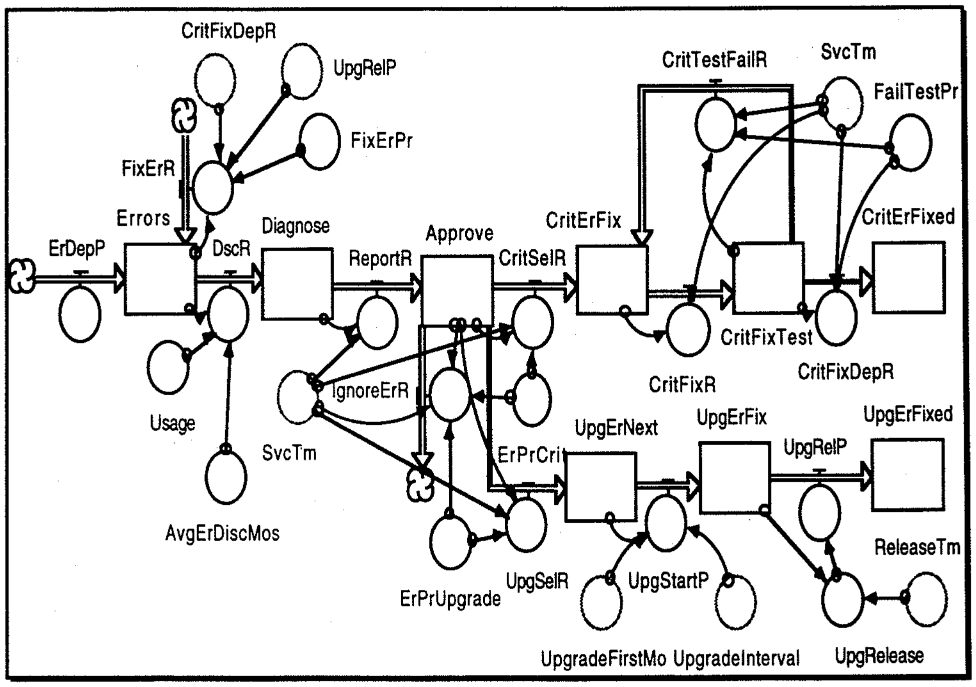


Figure 4. Error Fix Process

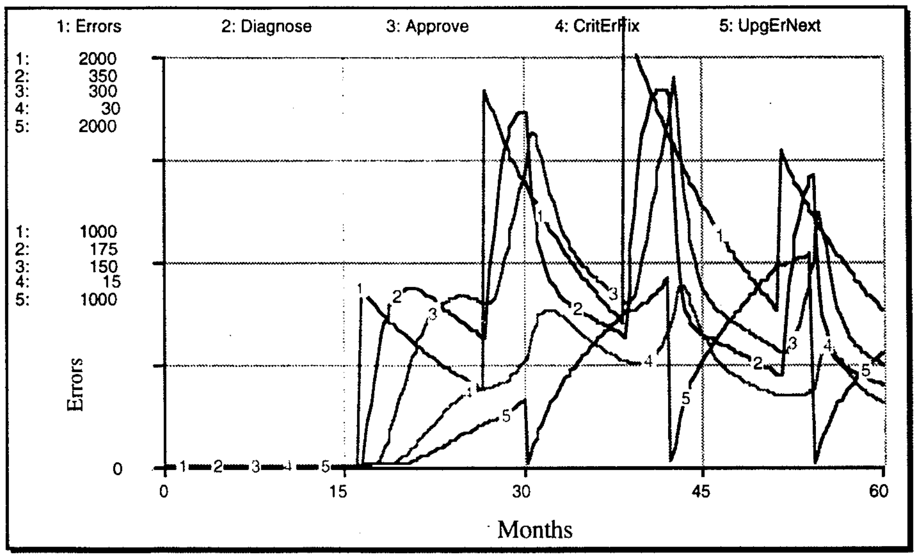


Figure 5. Sample Error Fix Process Run

Decision parameters

Figure 7 shows the decision parameters in our life cycle model and their default values. These are shown in the S**4 cockpit that provides the user game interface. A model run can be stepped to any time period and stopped. The user can then check any of the output reports, graphs or tables. In addition, the analyst can change any of the decision parameter values prior to restarting the simulation. The decision variables form the basis for parametric sensitivity analysis.

The decision parameters are divided into control, maintenance process, ICASE tool, environment, staffing, monthly cost, and functional relationships categories. Functional relationships are specified as power curves. This provides easy calibration of relationships to user organization data. Monthly cost values are specific to a customer location. Environment parameters relate to the specific software platform including its: investment, operations cost, remaining life, and software impact on mission effectiveness.

Figure 8 shows a sample animated report which summarizes both the previously described error fix and upgrade processes. In addition, cost/value, staff, error, and functionality performance measures are given in separate rectangles. The cost/value summary display gives the monthly spending rate, cumulative cost from the start of the simulation, present platform investment, cumulative value of maintenance and upgrade actions, and the benefit-cost ratio.

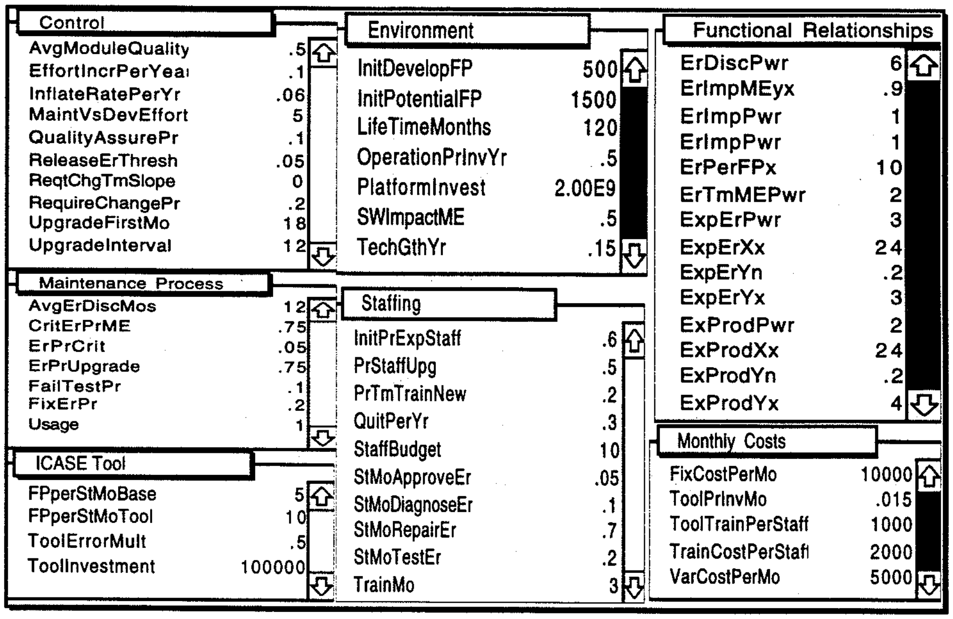
The staff summary display gives the present number of new and experienced staff. The labor productivity is given in terms of function points per staff month. Productivity is based on staff experience and tools employed. The average module quality level reported is a function of staff experience, tools employed and the proportion of time allocated to quality assurance. The service time is the average time for the error fix process execution. Staff are allocated to the error fix activities in proportion to workload so perfect load balance is achieved.

The errors summary display shows the present number of undiscovered errors, and the proportion of critical and upgrade errors fixed. The error impact on mission effectiveness is shown as a multiplier (unity for no errors). Finally the monthly cost of any remaining undiscovered errors is given.

The functionality summary display gives the present potential function points and deployed system function points. Potential function points is the size of the ideal software support package at the present technology level. The function ratio is the present ratio of system function points divided by potential function points divided by the ratio at the start of the simulation. If a lower proportion of potential function points are implemented, platform mission effectiveness suffers due to lower software functionality.

Scenario sensitivity analysis

Program runs were executed to investigate the impact of individual parameter changes when all other parameters were set to their base default values. The benefit-cost ratio was evaluated with respect to the platform, environment, tool and requirement change parameters. Other measures were calculated for the same parameters but space limitations preclude their presentation. The parameters were set at low, middle (base), and high values.



Control

AvgModuleQuality	.5
EffortIncrPerYeai	.1
InflateRatePerYr	.06
MaintVsDevEffort	5
QualityAssurePr	.1
ReleaseErThresh	.05
ReqtChgTmSlope	0
RequireChangePr	.2
UpgradeFirstMo	18
UpgradeInterval	12

Maintenance Process

AvgErDiscMos	12
CritErPrME	.75
ErPrCrit	.05
ErPrUpgrade	.75
FailTestPr	.1
FixErPr	.2
Usage	1

ICASE Tool

FPperStMoBase	5
FPperStMoTool	10
ToolErrorMult	.5
ToolInvestment	100000

Environment

InitDevelopFP	500
InitPotentialFP	1500
LifeTimeMonths	120
OperationPrInvYr	.5
PlatformInvest	2.00E9
SWImpactME	.5
TechGthYr	.15

Staffing

InitPrExpStaff	.6
PrStaffUpg	.5
PrTmTrainNew	.2
QuitPerYr	.3
StaffBudget	10
StMoApproveEr	.05
StMoDiagnoseEr	.1
StMoRepairEr	.7
StMoTestEr	.2
TrainMo	3

Functional Relationships

ErDiscPwr	6
ErImpMEyx	.9
ErImpPwr	1
ErImpPwr	1
ErPerFPx	10
ErTmMEPwr	2
ExpErPwr	3
ExpErXx	24
ExpErYn	.2
ExpErYx	3
ExProdPwr	2
ExProdXx	24
ExProdYn	.2
ExProdYx	4

Monthly Costs

FixCostPerMo	10000
ToolPrInvMo	.015
ToolTrainPerStaff	1000
TrainCostPerStafi	2000
VarCostPerMo	5000

Figure 7. S**4 Decision Parameters

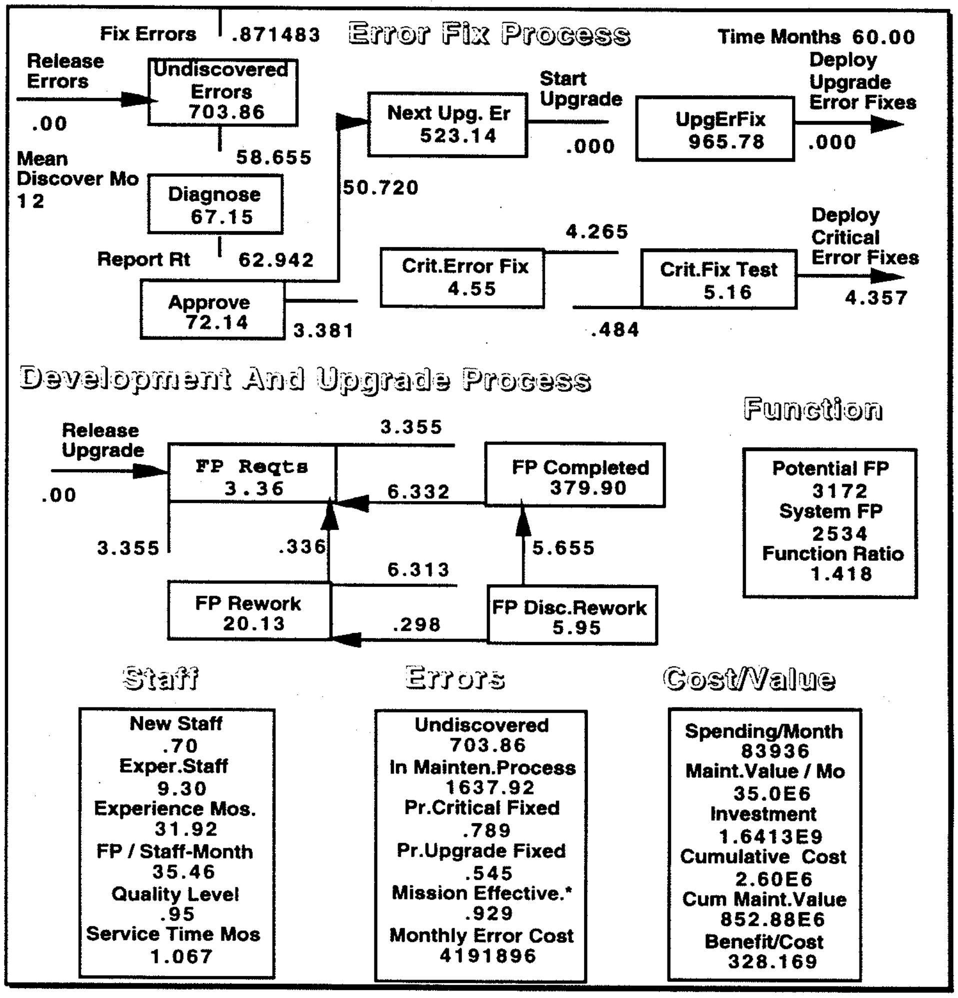


Figure 8. S**4 Animated Report

Platform parameters are presented in figure 9. The first parameter is unamortized platform investment at the start of the simulation. It ranges from .5 billion dollars to 3.5 billion dollars with a base value of 2 billion dollars. It has a significant impact on benefit-cost performance. The next parameter is operations and maintenance costs per year as a percentage of platform investment. It ranges from 25% to 75% with a baseline value of 50%, and also is significant. The third significant parameter is the software impact on mission effectiveness which ranges from 25% to 75% with a base value of 50%. The fourth parameter is the remaining platform life over which the platform investment is amortized. Shorter life means a larger value per year and hence more benefit from mission effectiveness improvements. The next to last parameter is the initial development function points. Lower initial function point deployment provides a larger potential gain from maintenance and upgrade activity. The last parameter is the proportion of staff allocated to upgrade activities. At the baseline operating point more personnel should be allocated to upgrades than to fixing errors.

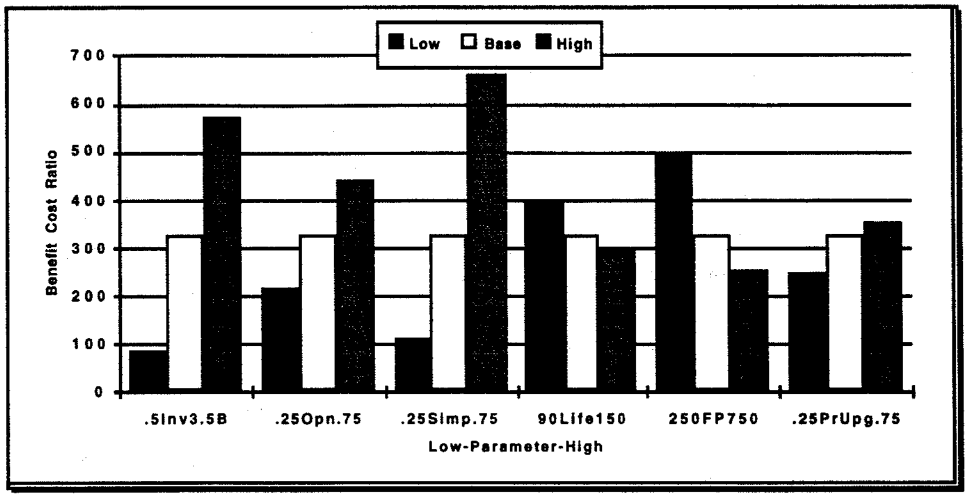


Figure 9. Platform Parameter Sensitivity Analysis

Environment parameter sensitivity analysis is presented in figure 10. Increasing staff from 5 to 15 leads to less marginal benefit-cost return. In an absolute sense, significant value is received ($658M to $900M) though at a cost increase ($1.58M to $3.62M). Increasing the annual personnel quit rate from 15% to 45% reduces the benefit cost ratio. This is due to lower inexperienced labor productivity, higher software error rates, and greater training costs. Increasing average module quality improves the benefit-cost ratio. Increasing the average months to discover an error from 6 to 18 months reduces the overall benefit-cost ratio. Increasing the proportion of requirements changed during the development or upgrade process from 10% to 30% reduces the benefit-cost ratio. Finally increasing the error threshold for the earlier release with more errors proves to be more cost effective. The benefits from earlier added functionality and upgrade error fixes more than compensates for the greater cost of fixing errors in maintenance.

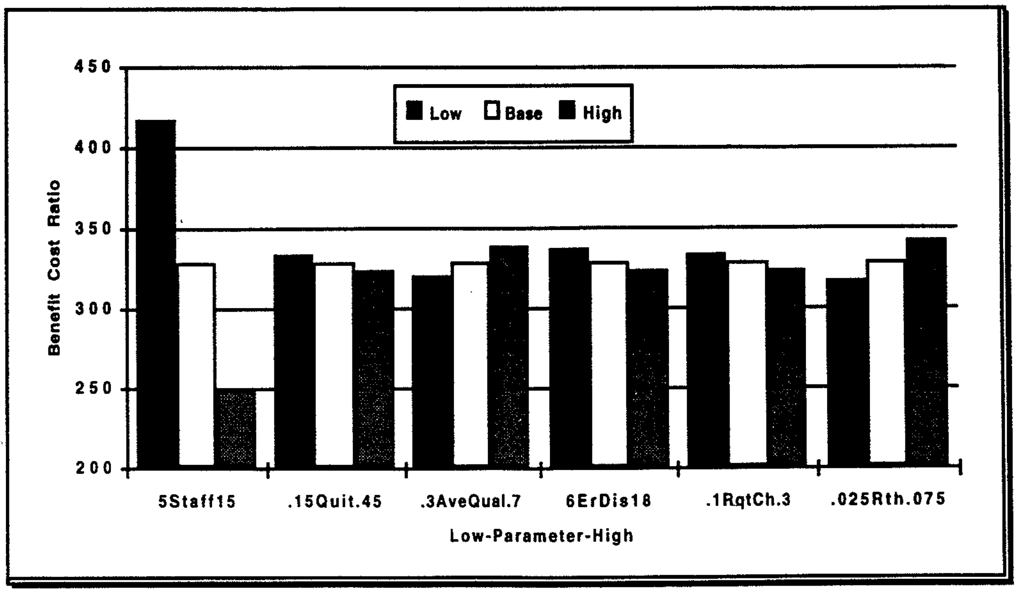


Figure 10. Environment Parameter Sensitivity Analysis

Conclusions

The dynamic software life cycle model presented in this paper is in an early stage of development. S**4 array capabilities will be exploited to add dynamic uncertainty analysis, and multiple software platforms competing for maintenance center resources in the next model release. Data on key model parameters is being collected from various organizations. This data will be used to calibrate the model to the organization's environment

This system dynamics model spans the entire software life cycle. Our model provides a structure for a software metrics collection program. A major contribution of this model is that it can be used to evaluate quantitatively several different software life cycle strategies.

References

Cooper,D.E.1993. *Test And Evaluation: DOD Has Been Slow in Improving Testing of Software-Intensive Systems.* September 29,1993 GAO report
B-253411, United States General Accounting Office, Washington, D.C.

Abdel-Hamid,T.,Madnick,S.E.1990.*Software Project Dynamics An Integrated Approach.* Prentice Hall, Englewood Cliffs, New Jersey

Nguyen,N.,Smith,B.,Vidale,F.1993. *Death Of A Software Manager: How To Avoid Career Suicide Through Dynamic Software Process Modeling.* American Programmer May 1993 Vol.6no.5

Cooper,K.G.,Mullen,T.W.1993. *Swords and Plowshares: The Rework Cycles of Defense and Commercial Software Development Projects.* American Programmer May 1993 Vol.6 No.5

Jones,C.1991. *Applied Software Measurement: Assuring Productivity and Quality.* McGraw-Hill,Inc. New York.

Richmond,B. et al.1991.*i think User's Guide.* High Performance Systems Inc.,Hanover, New Hampshire

Diehl,E.1992.*S**4 The Strategy Support Simulation System.* Microworlds Inc. Boston, Massachusetts