

Leading and Lagging Indicators of Project Schedule Problems

by

David Malloy, Ph.D., PMP

Deloitte Consulting, LLP

dmalloy@deloitte.com

610-724-1031

Introduction

Schedule variance (SV) and schedule performance index (SPI) are common metrics used to manage large development programs, such as in the aerospace and defense (A&D) industry. However, SV and SPI are lagging indicators of schedule performance, often with significant delay due to data gathering, compilation and reporting. The performance snapshot given by SV and SPI can be several weeks old on some large programs. Any problems indicated will have potentially gotten worse.

Also, SV and SPI reflect the perceived progress as reported by the project management team. Unfortunately, the actual progress is likely to be less, due to hidden errors in work already finished. When the problems are uncovered, reported progress must be reduced. Again, SV and SPI alone are not effective management metrics, reflecting only the known, past problems.

To address these issues, leading indicators of schedule performance must augment SPI and SV. For example, on large A&D programs with significant software development efforts, potential leading indicators include time spent preparing for code reviews, the number of peer reviews held, and the number of prototypes tested. Also, key metrics to encourage the early discovery of hidden rework must be employed.

This paper investigates proactive and leading indicators and develops a simulation model to illustrate their use. The simulation model of the project can help select unique, non-redundant metrics to better manage the program and control the schedule. Use of the simulation model during project planning can also improve the duration estimates and overall program estimates.

Aerospace & Defense Software Development

Managing large A&D product development programs is a significant challenge, given the technological risks, large investments and high level of public and private scrutiny. As the GAO noted [1], “reviews over the past 30 years have found consistent problems with ... acquisitions such as cost increases, schedule delays, and performance shortfalls.” Sophisticated software is typically a critical component in most, if not all A&D systems. The A&D industry boasts impressive software development prowess, with Lockheed Martin claiming to employ more software engineers than Microsoft. Yet, software development is particularly vexing for A&D management, with one survey [2] indicating that defense software project quality was half that of commercial projects, with a 34% versus 68% first pass acceptance rate, respectively. Some of

this difference can be attributed to the changes in requirements often imposed by the customer (e.g. DoD) or the inherently leading edge nature of the projects. However, given the national security importance and amount of public resources devoted to defense product development, improvement is clearly necessary.

A&D software development programs follow any number of management approaches, ranging from the more traditional waterfall method to more recently, spiral development. For large cost-plus defense programs, customers generally require rigorous schedule control and earned value management [3]. Schedule performance reporting will involve calculating the current schedule variance (SV) and schedule performance index (SPI) relative to the schedule baseline. Typical calculations are shown below:

$$SV\% = \frac{EarnedValue - PlannedValue}{PlannedValue} 100 \qquad SPI = \frac{EarnedValue}{PlannedValue}$$

Or using the Department of Defense (DoD) preferred terms for Earned Value (Budgeted Cost of Work Performed) and Planned Value (Budgeted Cost of Work Scheduled)

$$SV\% = \frac{BCWP - BCWS}{BCWS} 100 \qquad SPI = \frac{BCWP}{BCWS}$$

While these calculations are straightforward, reporting these metrics to management occurs after delays of anywhere from one week to one month, depending on the size of the program, difficulty in assessing performance, and ease of use of supporting project information and reporting systems. These inherent delays limit the day-to-day usefulness of these schedule metrics.

Furthermore, the assessment of earned value is based on the project management team's perception of current progress. Of course, this perception is fallible, and generally assumes that the work accomplished to date does not contain substantial errors. As noted above, A&D software development quality can be quite low. Project managers typically do not understand the real quality level of their software teams' output, until the problems are uncovered downstream in initial testing or integration.

While most experienced project managers would acknowledge that their estimates of current progress are optimistic and do not discount for undiscovered rework, few programs plan adequately for rework. Invariably, these rework activities are underestimated, with adverse impacts on schedule and cost performance. Indeed, as numerous researchers have observed, rework on software development and other complex projects is a major contributor to poor project performance [4, 5].

Given the delays in reporting and optimistic progress estimates, the earned value schedule metrics SV and SPI are at best lagging indicators of performance, and at worst fundamentally inaccurate. SV and SPI can be useful reporting metrics, particularly at an aggregate level. But for the software manager or team leader trying to gage his or her team's performance and more

importantly guide their daily activity, more timely and leading performance indicators are needed. Because of the corrosive effect that hidden errors have on current and downstream performance, managers should focus on metrics that highlight meaningful contributions to improved quality. For example, software teams employing code peer reviews can track the number of reviews held or even the time spent preparing for the reviews. Of course, these measures are only effective if by focusing on them, the actual quality of the product improves.

Simulation Model

To better understand the interaction of team quality and productivity on schedule performance as measured by SV and SPI, and to investigate the potential benefits of other project metrics, a simulation model was developed of a representative aerospace and defense software development program. The notional program is for development of software for an aircraft system, using a simplified composite of past programs based on the author’s experience in industry and consulting. The project consists of three teams, the first synthesizing and defining the software requirements, derived from the aircraft and particular system requirements. The second team takes the software requirements and builds software code. The final team tests the software against the requirements. Of course, this model neglects the many other interfaces the project team would have on a large aircraft development program (e.g. other systems, hardware, integration, ...) but it is useful for illustrating the principal flow of work on a project.

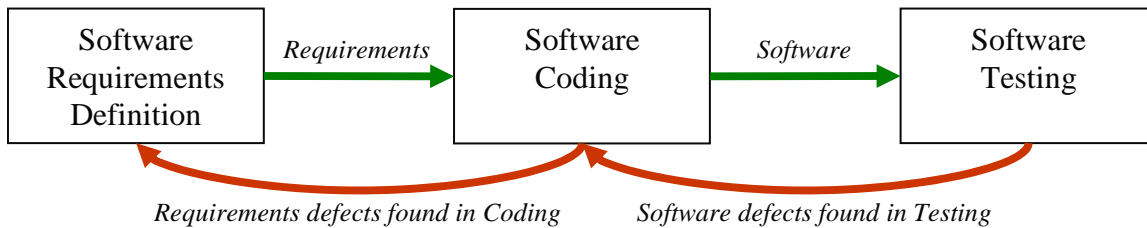


Figure 1: *Notional Program Model*

As shown above in Figure 1, the work passes from the requirements team to the coding team to the testing team. Problems found by a downstream team are sent back to the upstream team. Any undiscovered errors passed from one team to another cause significant disruption. To avoid sending errors downstream, each team employs internal checks to uncover errors before handing off the work product. Even better, teams use proactive techniques such as peer reviews prior to detail coding to prevent errors from occurring.

Figure 2 depicts the model simulation building block that captures a team’s scheduled work plan and the actual flow of work. The upper part of the figure shows the ideal flow of tasks as scheduled in the project plan, moving from the *Scheduled Tasks to be Done* to *Scheduled Tasks Done* based on nominal team productivity and a schedule cushion or “float” factor. The lower part of the diagram tracks the progress in the “real” world, with most of the tasks successfully ending up in *Actual Tasks Done*, with the remainder in *ReWork Tasks*, with the split based on the team’s quality. As items needing rework are discovered through internal testing or review, they

are added back to *Actual Tasks to be Done*, to cycle through again. The team’s progress is perceived to be the tasks completed, including the undiscovered rework. After a delay, the team provides its performance status relative to the schedule baseline. If the team reports that it is falling behind schedule, pressure begins to build to increase its output. This schedule pressure increases its productivity, but decreases its quality – “haste makes waste.” This model incorporates several of the concepts described in project management modeling literature [4, 5, 6], tailored to reflect the author’s program management experience and to facilitate investigation into more effective metrics to augment traditional earned value reporting.

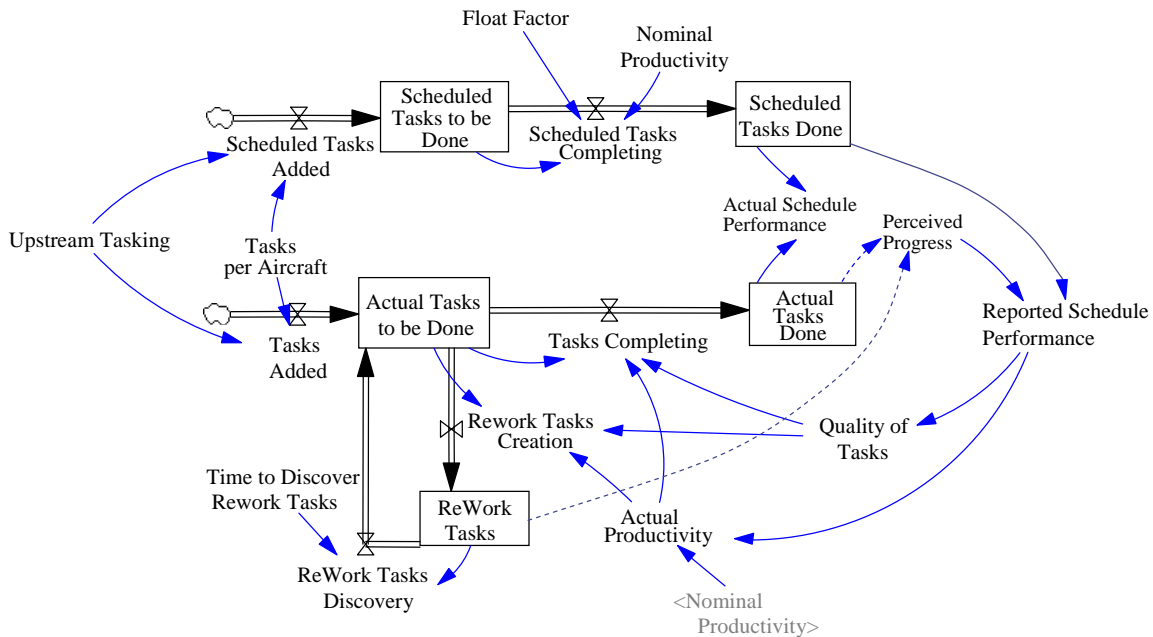


Figure 2: *Model of Team Internal Activities*

The project simulation model of all three teams’ efforts (requirements, coding and testing) is described in detail in the Appendix, and was implemented using Vensim, a popular system dynamics software package [7]. System dynamics is a powerful modeling approach used to analyze and address a wide range of scenarios in business, from strategic to operational issues [8, 9, 10]. Project management is one of the most prevalent applications of systems dynamics.

Simulation Results

To investigate the issues regarding the effectiveness of SV and SPI as project management metrics, the three team software development model was developed, reflecting a one-year project. The requirements team creates 10,000 detailed software requirements. The software coding team translates those requirements into 100,000 lines of code. The testing team then runs 2,500 tests to complete the project. The baseline model generates the project plan based on each team’s nominal productivity with a 5% schedule cushion or buffer.

The software requirements team’s effort is shown below in Figure 3. The planned schedule for requirements definition (blue line) shows completion during the twelfth month. The team’s perception of the requirements defined (red line) lags the plan, but exceeds their actual

accomplishments, shown in green. The difference between the team's perception and reality are the requirements needing rework, shown in gray.

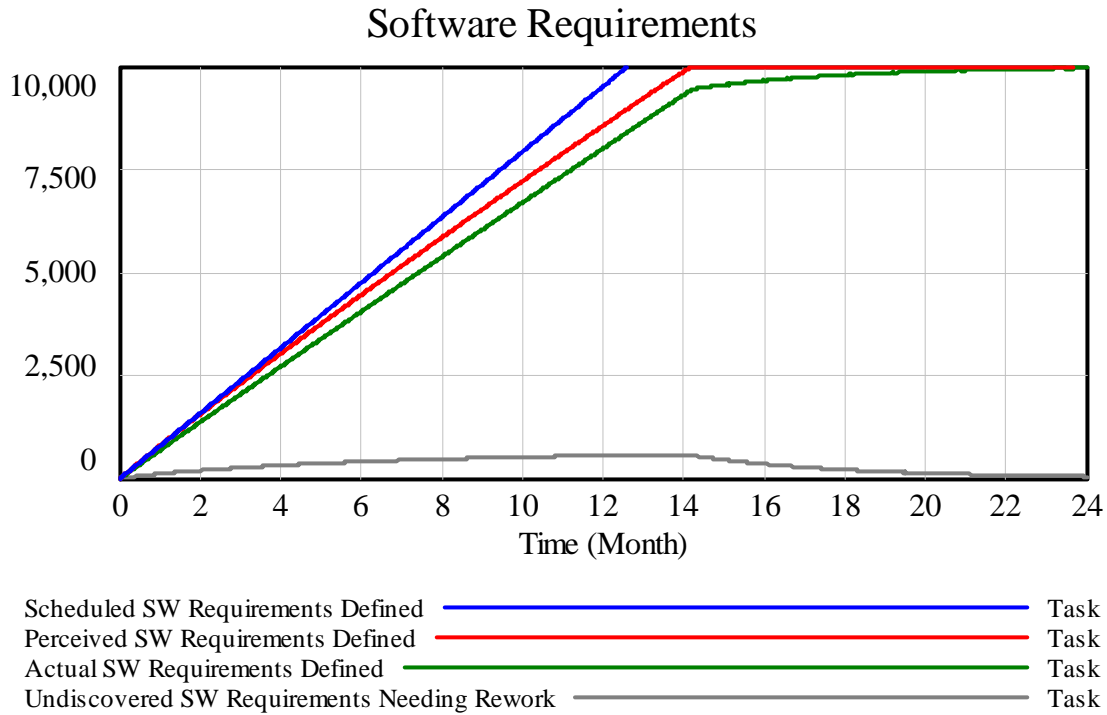


Figure 3: *Software Requirements Team Effort*

The undiscovered rework, along with the reporting delays, causes the reported SPI and SV metrics to significantly misrepresent the true schedule performance of the requirements team, as seen in Figures 4 and 5. The blue line is the status that the team reports to senior program management, while the red line represents the true situation.

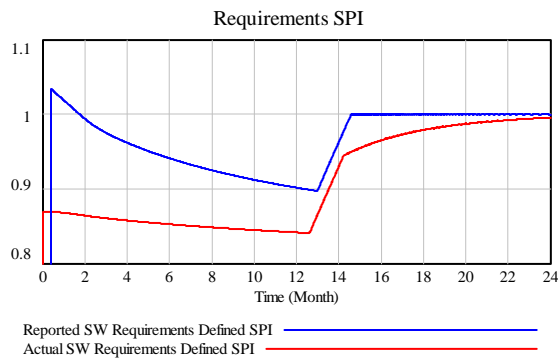


Figure 4: Software Requirements SPI

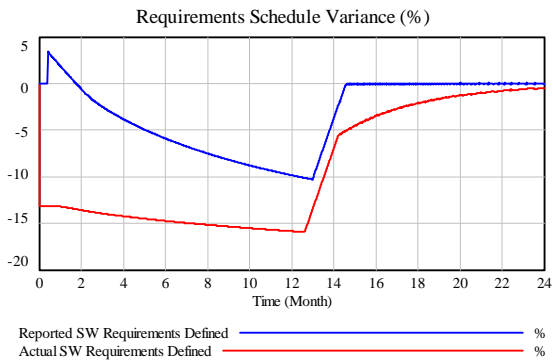


Figure 5: Software Requirements SV

The software coding team's effort exhibits similar characteristics, with undiscovered rework clouding perception of actual performance, and reporting lags further exacerbating SPI effectiveness.

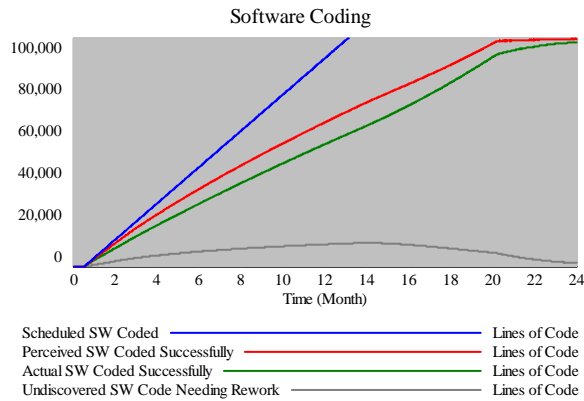


Figure 6: Software Coding Team Effort

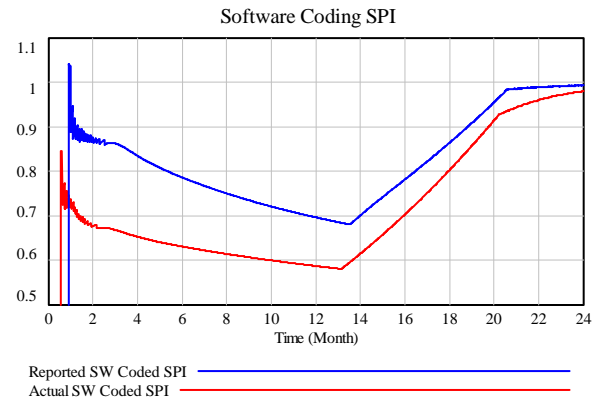


Figure 7: Software Coding SPI

As the software coding team realizes they are behind schedule, productivity increases, with a partially offsetting decrease in quality.

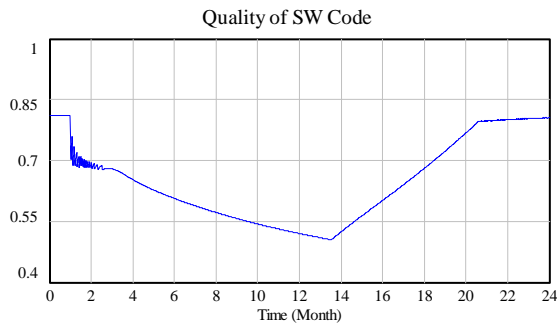


Figure 8: Quality of SW Code (1=100%)

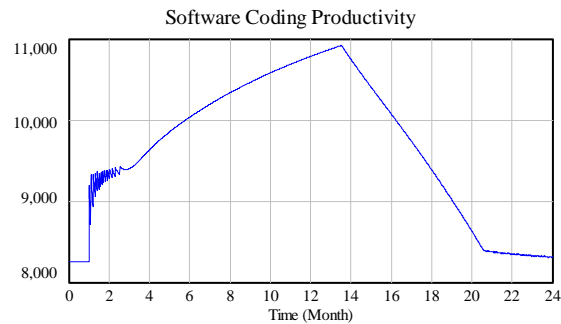


Figure 9: Coding Productivity (Lines/Month)

Due to the poor performance in the requirements and coding phases, the testing team falls significantly behind, finally finishing the project nearly one year late. While these results are dramatic, they are representative of schedule overruns common in A&D software development projects [2, 4]. In fact, the model is conservative, with the coding quality used in the simulation exceeding the effective quality demonstrated by the average A&D software project.

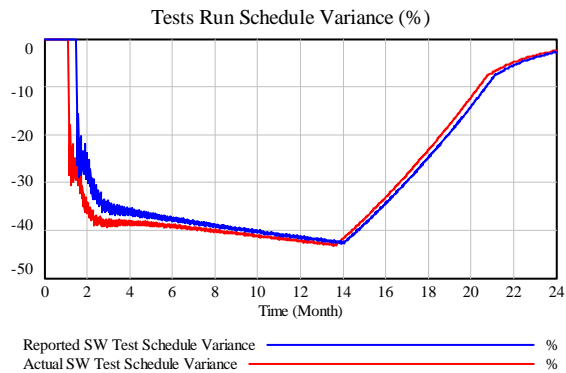


Figure 10: Testing Schedule Variance

Policies to Improve Performance and Supporting Metrics

The simulation model enables evaluation of project management policies to counter poor schedule performance. Also, the model allows selection of appropriate metrics to encourage improved project performance. Indeed, system dynamic models are valuable project management tools during all program phases, from bid and proposal, through execution, to capturing lessons learned.

In this case, poor quality is driving much of the schedule challenge. Therefore, to augment the standard earned value metrics of SV and SPI, other performance measures will be used, with the goal of improving quality, which ultimately leads to better schedule execution. One approach is to track the number of peer reviews or more formal appraisals held within each team during the course of the project. For example, probing, detailed examination of portions of the work-in-process by a council of peers fosters an open exchange of ideas and helps mentor more junior staff members. If conducted effectively, such reviews can improve overall team quality.

However, these meetings require an investment in staff time for preparation, discussion and resolution of issues. Therefore, productivity will suffer as a function of the number of meetings held. The tradeoff between quality and productivity can be readily analyzed using the simulation model, allowing the project management policy to be fine tuned. For example, several different review schedules were considered, from 1 every two months (baseline) up to 4 reviews per month.

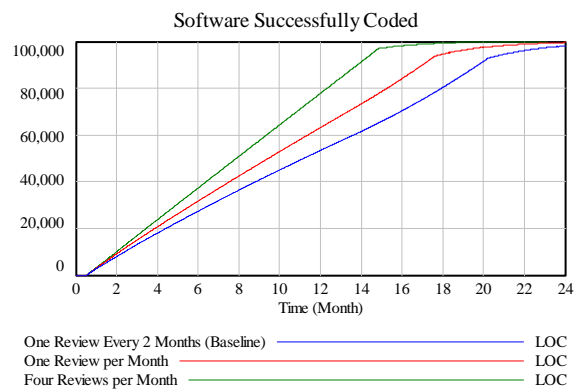


Figure 11: *SW Completed with Peer Review Tracking (Lines of Code)*

The corresponding software coding quality and productivity are shown in Figures 12 and 13, highlighting the effect of increased frequency of peer reviews. Clearly, the benefits of the increase in quality outweigh the productivity penalty. With four reviews per month, the project finishes much earlier than the baseline simulation.

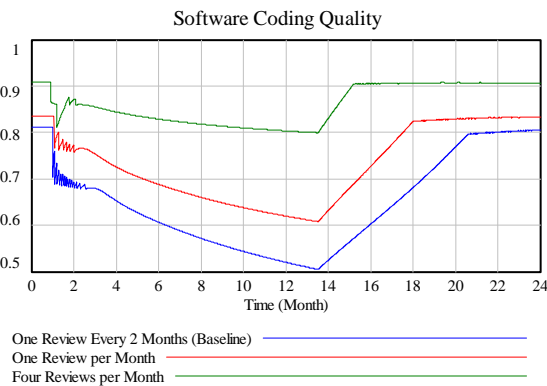


Figure 12: *Software Quality (1=100%)*

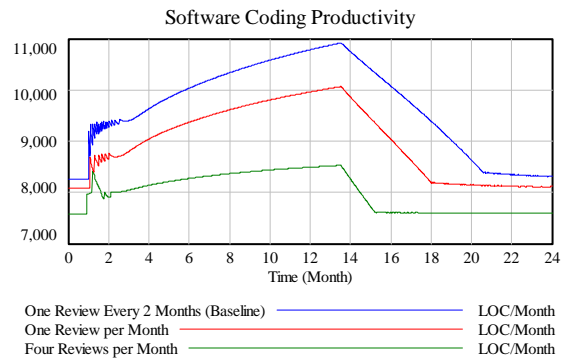


Figure 13: *Software Productivity*

Rather than have a set number of review meetings each month, an alternative policy is to schedule the peer evaluations based on the amount of internal (and external) rework discovered. This enables the project manager to quickly respond to rising rework, while maintaining productivity early in the project. As seen in Figures 14 through 16, the project keeps a high quality level with sufficient productivity to allow the project to finish nearly on time.

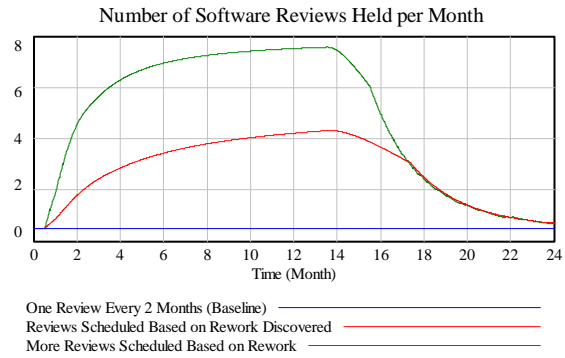


Figure 14: Software Code Reviews

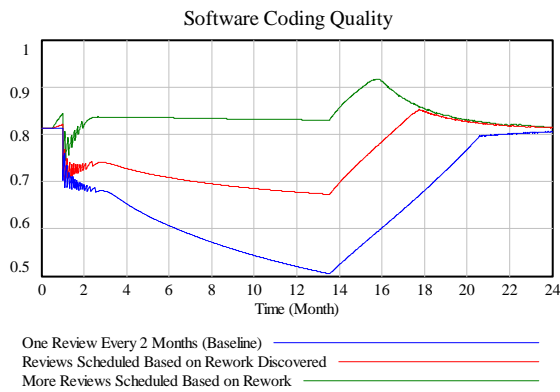


Figure 15: Software Quality (1=100%)

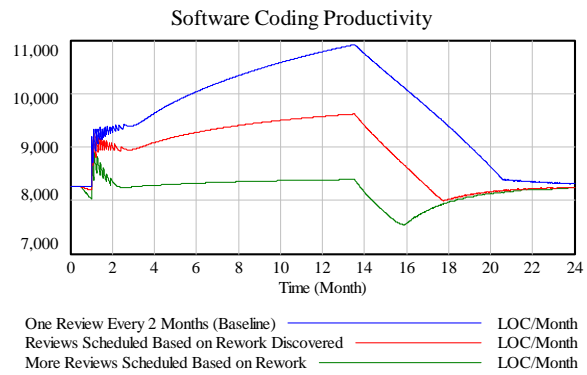


Figure 16: Software Productivity

Finally, another policy change may mitigate the schedule pressure felt by the teams as unplanned rework causes them to fall behind the plan. Simply allowing more of a schedule buffer enables teams to focus on producing quality work, and avoids the detrimental effects of schedule pressure. As Figures 17 through 19 below illustrate for the software coding team, additional schedule allowance results in finishing the project earlier, a somewhat counterintuitive result. However, when considering the higher quality exhibited by the coding team (and all other teams), the project avoids the “haste makes waste” trap. More time is spent delivering good code, and less is spent creating errors needing rework.

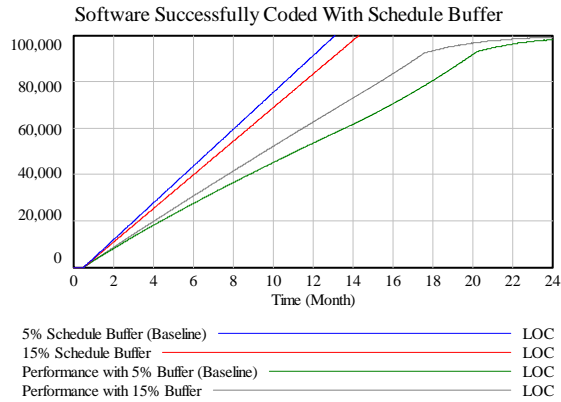


Figure 17: Scheduled and Actual SW Coded

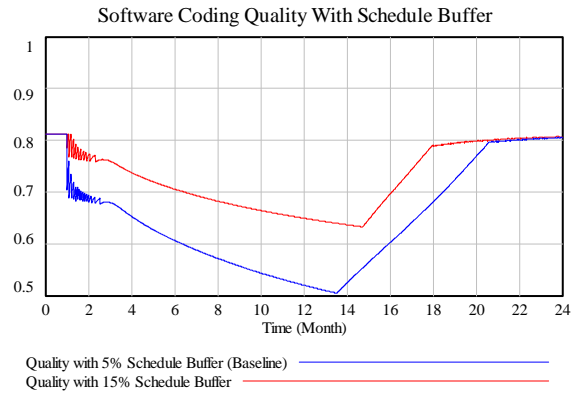


Figure 18: Coding Quality (1=100%)

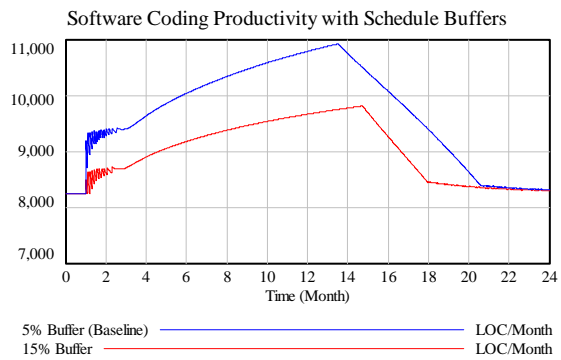


Figure 19: Coding Productivity

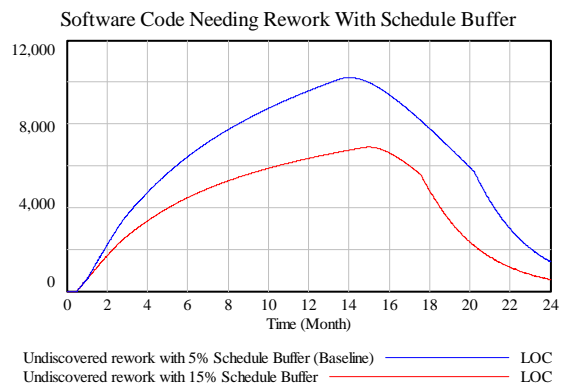


Figure 20: Coding Rework

Conclusion

The schedule metrics SV and SPI are inherently inaccurate and significantly lag the true project situation. To better manage project schedule execution, more leading indicators should be used to augment SV and SPI. This paper focused on several measures that highlight activity that contributes to product quality, such as peer reviews in a software development project. A simulation model facilitated evaluating different project management policies and supporting metrics. Several quality indicators that lead overall schedule performance were found to allow more proactive project guidance. Several policies were assessed and show significant promise for project management improvement.

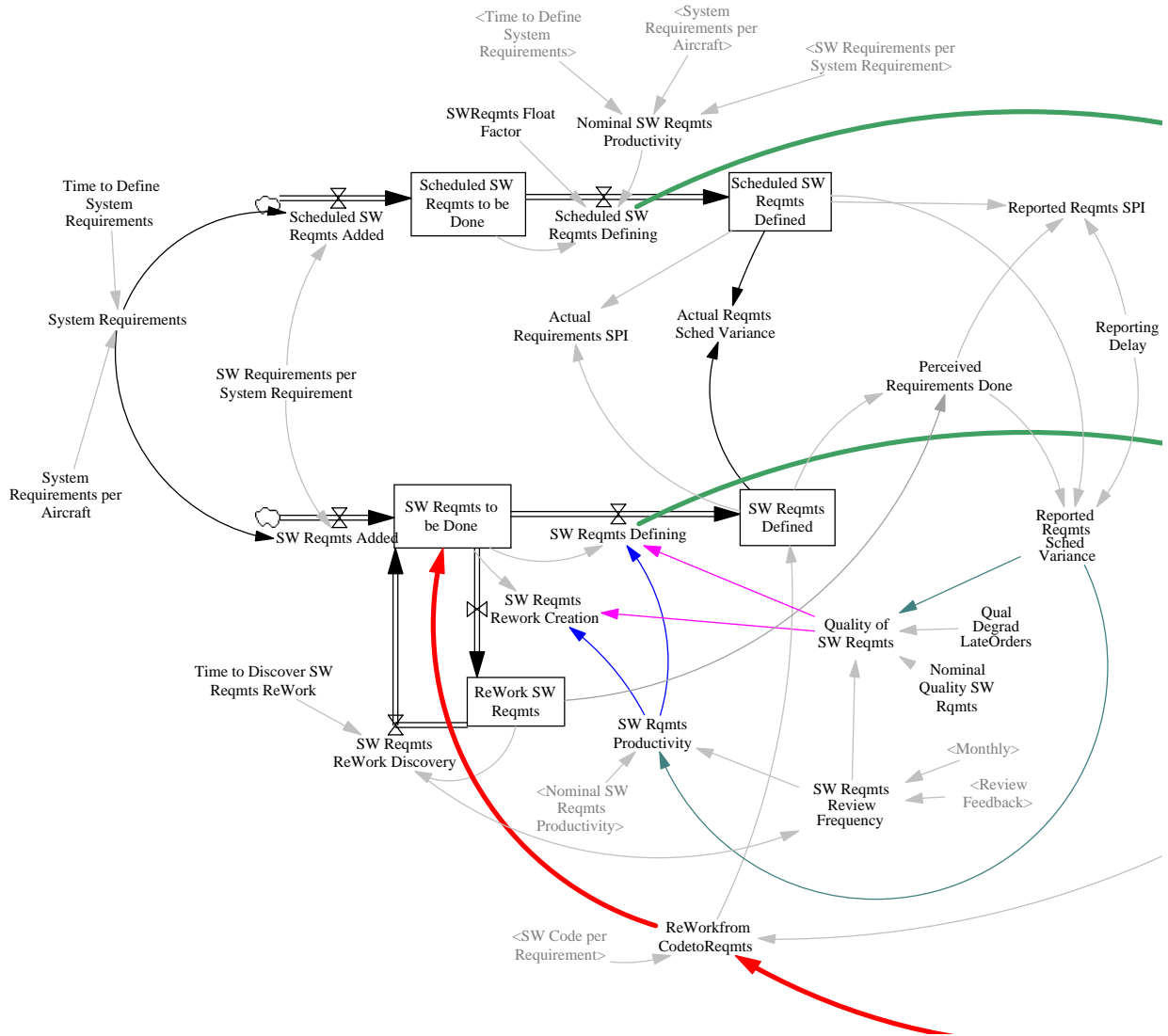
Further development of the simulation model to incorporate staffing and cost information would enable investigation into the effectiveness of other earned value metrics, such as cost variance and the cost performance index.

References

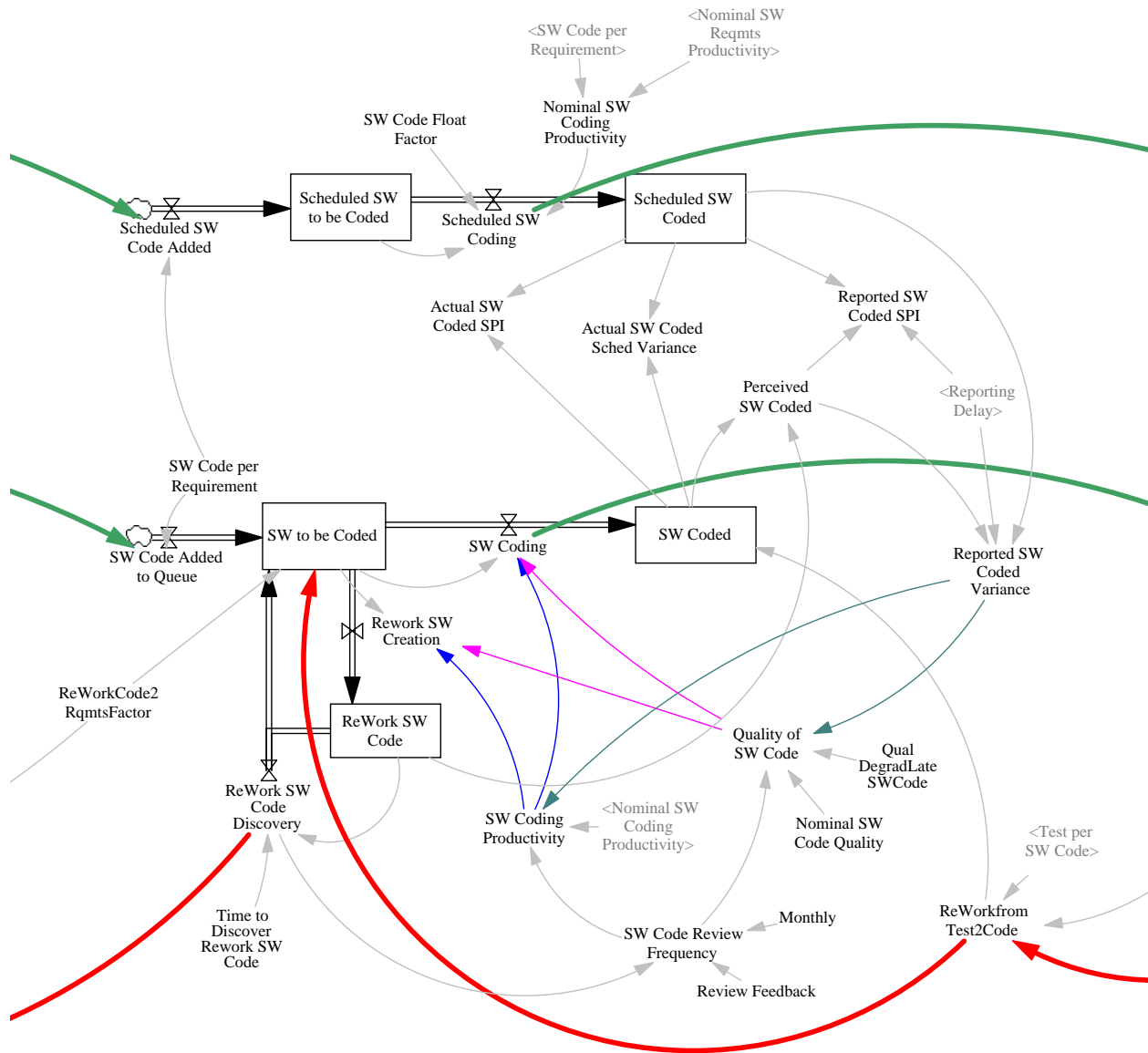
- [1] Assessments of Selected Major Weapon Programs, Government Accounting Office Report GAO-05-301, March 2005.
- [2] Cooper, K.G. and Mullen, T.W., Swords & plowshares: the rework cycles of defense & commercial software development projects, *PA Consulting Report*, PA Consulting Group, Cambridge, Massachusetts 02142.
- [3] *U.S. Department of Defense Extension to: A Guide to the Project Management Body of Knowledge (PMBOK® Guide) 1st Edition*, Defense Acquisition University Press, Fort Belvoir, VA, June 2003.
- [4] Lyneis, J.M., Cooper, K.G. and Els, S.A., Strategic management of complex projects: a case study using system dynamics, *System Dynamics Review*, Volume 17 Number 3 Fall 2001
- [5] Abdel-Hamid, T.K., and Madnick, S.E., Lessons learned from modeling the dynamics of software development, *Communications of the ACM*, December 1989 Volume 32 Number 12
- [6] Lyneis, J.M., Course notes from 15.982.c, *MIT's Advanced Study Program in System Dynamics: Project Dynamics*, Fall 2002.
- [7] Ventana Systems, Inc., Harvard, MA 01451. URL: www.Vensim.com
- [8] Sterman, J. D., *Business Dynamics: Systems Thinking and Modeling for a Complex World*, Irwin McGraw Hill, 2000.
- [9] Warren, K., Dynamics of Strategy, *Business Strategy Review*, 1999, Volume 10 Issue 3.
- [10] Malloy, D., Modeling the life cycle cost impact of product development decisions in an aerospace supply chain, *System Dynamics Conference*, New York, July, 2004.

Appendix – Model Description

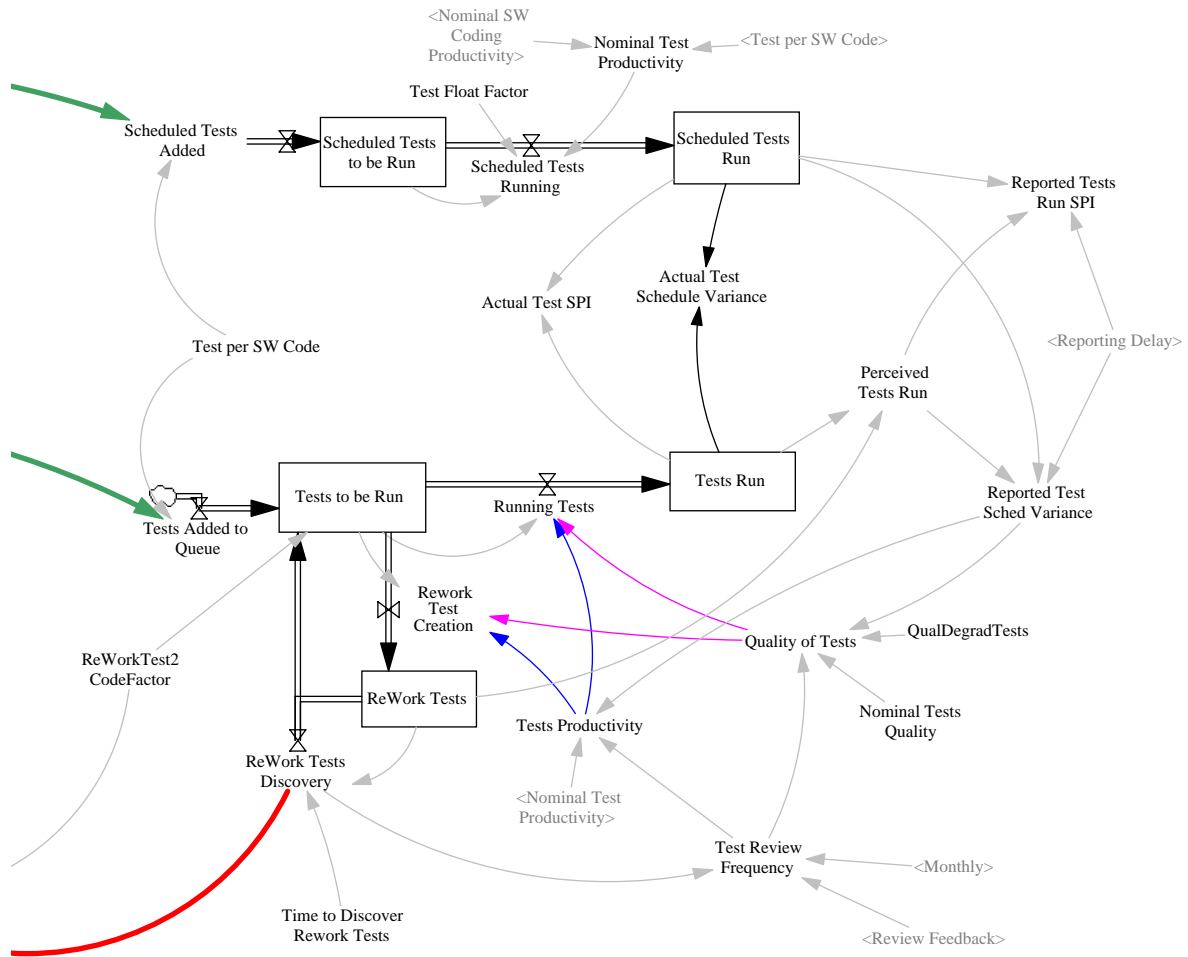
Software Requirements Definition



Software Coding



Software Testing



Vensim Model Equations

SW Reqmts Defining=
 IF THEN ELSE(SW Reqmts to be Done>0, SW Rqmts Productivity , 0)*Quality of SW Reqmts
 ~ Task/Month

Nominal Test Productivity=
 Nominal SW Coding Productivity*Test per SW Code
 ~ Task/Month

Nominal SW Coding Productivity=
 Nominal SW Reqmts Productivity*SW Code per Requirement
 ~ Task/Month

ReWork Tests Discovery=
 ReWork Tests/Time to Discover Rework Tests
 ~ Task/Month

Nominal SW Reqmts Productivity=
 SW Requirements per System Requirement*System Requirements per Aircraft/Time to Define System Requirements
 ~ Task/Month

Reported Reqmts SPI= DELAY FIXED (
 Perceived Requirements Done/(Scheduled SW Reqmts Defined+1e-006) , Reporting Delay , 0)

Reported SW Coded SPI= DELAY FIXED (
 Perceived SW Coded/(Scheduled SW Coded+1e-006) , Reporting Delay , 0)

Scheduled SW Coding=
 IF THEN ELSE(Scheduled SW to be Coded>0 , Nominal SW Coding Productivity/SW Code Float Factor
 , 0)
 ~ Task/Month

Reported Test Sched Variance= DELAY FIXED (
 (Perceived Tests Run-Scheduled Tests Run)/(Scheduled Tests Run+1e-007)*100 , Reporting Delay , 0)

Reported Tests Run SPI= DELAY FIXED (
 Perceived Tests Run/(Scheduled Tests Run+1e-006) , Reporting Delay , 0)

Scheduled SW Reqmts Defining=
 IF THEN ELSE(Scheduled SW Reqmts to be Done>0 , Nominal SW Reqmts Productivity/SWReqmts Float Factor, 0)
 ~ Task/Month

Monthly=
 0

Review Feedback=
 0

Quality of SW Code=
 Nominal SW Code Quality*max(0.3,(1+min(0,QualDegradLateSWCode*Reported SW Coded Variance\
)+0.03*SW Code Review Frequency))
 ~ Dmnl

Quality of Tests=
 Nominal Tests Quality*max(0.5,(1+min(0,QualDegradTests*Reported Test Sched Variance\
)+0.03*Test Review Frequency))
 ~ Dmnl

Test Review Frequency=
 0.5*(1+ReWork Tests Discovery/8*Review Feedback)+(1-Review Feedback)*Monthly
 ~ Task/Month

Tests Productivity=
 Nominal Test Productivity*max(0.5,(1-min(0,Reported Test Sched Variance*0.05)-0.02*Test Review Frequency\
))
 ~ Task/Month

Quality of SW Reqmts=
 max(min(0.99,Nominal Quality SW Rqmts*(1+min(0,QualDegradLateOrders*Reported Reqmts Sched Variance\
)+SW Reqmts Review Frequency*0.1)),0.4)
 ~ Dmnl

SW Code Review Frequency=
 0.5*(1+ReWork SW Code Discovery/400*Review Feedback)+(1-Review Feedback)*Monthly
 ~ Task/Month

Reporting Delay=
 0.375
 ~ Month

SW Rqmts Productivity=
 Nominal SW Reqmts Productivity*(1-min(0.01*Reported Reqmts Sched Variance,0)-0.03*SW Reqmts Review Frequency\
)
 ~ Task/Month

SW Coding Productivity=
 Nominal SW Coding Productivity*(1-min(0,0.01*Reported SW Coded Variance)-0.02*SW Code Review Frequency\
)
 ~ Task/Month

Reported SW Coded Variance= DELAY FIXED (
 (Perceived SW Coded-Scheduled SW Coded)/(Scheduled SW Coded+1e-007)*100 , Reporting Delay , 0)

SW Reqmts Review Frequency=
 0.5*(1+SW Reqmts ReWork Discovery/40*Review Feedback)+(1-Review Feedback)*Monthly
 ~ Task/Month

Reported Reqmts Sched Variance= DELAY FIXED (
 (Perceived Requirements Done-Scheduled SW Reqmts Defined)/(Scheduled SW Reqmts Defined)
 +1e-007)*100,Reporting Delay,0)
 ~ Percent

Actual Test SPI=
 Tests Run/(Scheduled Tests Run+1e-007)

Actual Requirements SPI=
 SW Reqmts Defined/(Scheduled SW Reqmts Defined+1e-007)

Actual SW Coded Sched Variance=
 (SW Coded-Scheduled SW Coded)/(Scheduled SW Coded+1e-007)*100
 ~ Task

Actual SW Coded SPI=
 SW Coded/(Scheduled SW Coded+1e-007)

Perceived Requirements Done=
 ReWork SW Reqmts+SW Reqmts Defined

Perceived Tests Run=
 ReWork Tests+Tests Run

Perceived SW Coded=
 ReWork SW Code+SW Coded

Actual Reqmts Sched Variance=
 (SW Reqmts Defined-Scheduled SW Reqmts Defined)/(Scheduled SW Reqmts Defined+1e-007)*100
 ~ Percent

Actual Test Schedule Variance=
 (Tests Run-Scheduled Tests Run)/(Scheduled Tests Run+1e-007)*100
 ~ Percent

Nominal Quality SW Rqmts=
 0.8
 ~ Dmnl

Nominal SW Code Quality=
 0.8
 ~ Dmnl

Nominal Tests Quality=
 0.95
 ~ Dmnl

QualDegradLateOrders=
 0.008

QualDegradLateSWCode=
 0.012

QualDegradTests=
 0.007

ReWork SW Code= INTEG (
 Rework SW Creation-ReWork SW Code Discovery,
 0)
 ~ Task

ReWork SW Code Discovery=
 ReWork SW Code/Time to Discover Rework SW Code
 ~ Task/Month

Rework SW Creation=
 IF THEN ELSE(SW to be Coded>0, SW Coding Productivity, 0)*(1-Quality of SW Code)
 ~ Task/Month

ReWork SW Reqmts= INTEG (
 SW Reqmts Rework Creation-SW Reqmts ReWork Discovery,
 0)
 ~ Task

Rework Test Creation=
 IF THEN ELSE(Tests to be Run>0, Tests Productivity, 0)*(1-Quality of Tests)
 ~ Task/Month

ReWork Tests= INTEG (
 Rework Test Creation-ReWork Tests Discovery,
 0)
 ~ Task

ReWorkCode2RqmtsFactor=
 0.1
 ~ Dmnl

ReWorkfromCodetoReqmts=

ReWork SW Code Discovery=ReWorkCode2RqmtsFactor/SW Code per Requirement
 ReWorkfromTest2Code=
 ReWork Tests Discovery*ReWorkTest2CodeFactor/Test per SW Code
 ReWorkTest2CodeFactor=
 0.8
 ~ Dmnl
 Running Tests=
 IF THEN ELSE(Tests to be Run>0, Tests Productivity , 0)*Quality of Tests
 ~ Task/Month
 Scheduled SW Code Added= DELAY FIXED (SW Code per Requirement*Scheduled SW Reqmts Defining , 0.5 , 0)
 ~ Task/Month
 Scheduled SW Coded= INTEG (Scheduled SW Coding, 0)
 ~ Task
 Scheduled SW Reqmts Added=
 System Requirements*SW Requirements per System Requirement
 ~ Task/Month
 Scheduled SW Reqmts Defined= INTEG (Scheduled SW Reqmts Defining, 0)
 ~ Task
 Scheduled SW Reqmts to be Done= INTEG (Scheduled SW Reqmts Added-Scheduled SW Reqmts Defining, 0)
 ~ Task
 Scheduled SW to be Coded= INTEG (Scheduled SW Code Added-Scheduled SW Coding, 0)
 ~ Task
 Scheduled Tests Added=
 DELAY FIXED(Test per SW Code*Scheduled SW Coding, 0.5, 0)
 ~ Task/Month
 Scheduled Tests Run= INTEG (Scheduled Tests Running, 0)
 ~ Task
 Scheduled Tests Running=
 IF THEN ELSE(Scheduled Tests to be Run>0 , Nominal Test Productivity/Test Float Factor , 0)
 ~ Task/Month
 Scheduled Tests to be Run= INTEG (Scheduled Tests Added-Scheduled Tests Running, 0)
 ~ Task
 SW Code Added to Queue=
 DELAY FIXED(SW Code per Requirement*SW Reqmts Defining, 0.5 , 0)
 ~ Task/Month
 SW Code Float Factor=
 1.05
 ~ Dmnl
 SW Code per Requirement=
 10
 ~ Dmnl
 SW Coded= INTEG (SW Coding-ReWorkfromTest2Code, 0)
 ~ Task
 SW Coding=
 IF THEN ELSE(SW to be Coded>0, SW Coding Productivity , 0)*Quality of SW Code
 ~ Task/Month
 SW Reqmts Added=
 System Requirements*SW Requirements per System Requirement
 ~ Task/Month
 SW Reqmts Defined= INTEG (SW Reqmts Defining-ReWorkfromCodetoReqmts, 0)
 ~ Task
 SW Reqmts Rework Creation=
 IF THEN ELSE(SW Reqmts to be Done>0, SW Rqmts Productivity, 0)*(1-Quality of SW Reqmts)
 ~ Task/Month

```

SW Reqmts ReWork Discovery=
  ReWork SW Reqmts/Time to Discover SW Reqmts ReWork
  ~ Task/Month

SW Reqmts to be Done= INTEG (
  SW Reqmts Added-SW Reqmts Defining-SW Reqmts Rework Creation+SW Reqmts ReWork Discovery\
  +ReWorkfromCodetoReqmts,
  0)
  ~ Task

SW Requirements per System Requirement=
  10
  ~ Task/Aircraft

SW to be Coded= INTEG (
  SW Code Added to Queue-SW Coding-Rework SW Creation+ReWork SW Code Discovery*(1-ReWorkCode2RqmtsFactor\
  )+ReWorkfromTest2Code,
  0)
  ~ Task

SWReqmts Float Factor=
  1.05
  ~ Dmnl

System Requirements=
  System Requirements per Aircraft/Time to Define System Requirements*PULSE(0,Time to Define System Requirements)
  ~ Aircraft/Month

System Requirements per Aircraft=
  1000

Test Float Factor=
  1.05
  ~ Dmnl

Test per SW Code=
  0.025
  ~ Dmnl

Tests Added to Queue=
  DELAY FIXED( SW Coding*Test per SW Code, 0.5, 0)
  ~ Task/Month

Tests Run= INTEG (Running Tests,0)
  ~ Task

Tests to be Run= INTEG (Tests Added to Queue-Running Tests-Rework Test Creation+ReWork Tests Discovery*(1-ReWorkTest2CodeFactor),0)
  ~ Task

Time to Define System Requirements=
  12
  ~ Month

Time to Discover Rework SW Code=
  2
  ~ Month

Time to Discover Rework Tests=
  0.25
  ~ Month

Time to Discover SW Reqmts ReWork=
  3
  ~ Month

*****
      .Control
*****_
      Simulation Control Parameters

FINAL TIME = 24
  ~ Month
  ~ The final time for the simulation.

INITIAL TIME = 0
  ~ Month
  ~ The initial time for the simulation.

SAVEPER =
  TIME STEP
  ~ Month [0.?]
  ~ The frequency with which output is stored.

TIME STEP = 0.015625
  ~ Month [0.?]
  ~ The time step for the simulation

```