NPS-AM-14-C11P13R01-051

PROCEEDINGS
OF THE
ELEVENTH ANNUAL ACQUISITION RESEARCH SYMPOSIUM

THURSDAY SESSIONS
VOLUME II

The Decline and Fall of Joint Acquisition Programs

Andrew Moore, Carnegie Mellon University
William Novak, Carnegie Mellon University
Matthews Collins, Carnegie Mellon University
Jay Marchetti, Carnegie Mellon University
Julie Cohen, Carnegie Mellon University

Published April 30, 2014

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

Panel 13. Acquisition Jointness and Its Effects

Thursday, May 15, 2014	
9:30 a.m. – 11:00 a.m.	**Chair: LTG Joseph L. Yakovac Jr.,** USA (Ret.), Naval Postgraduate School, former Military Deputy to the Assistant Secretary of the Army (Acquisition, Logistics, & Technology) ***The Decline and Fall of Joint Acquisition Programs*** Andrew Moore, Carnegie Mellon University William Novak, Carnegie Mellon University Matthews Collins, Carnegie Mellon University Jay Marchetti, Carnegie Mellon University Julie Cohen, Carnegie Mellon University ***The Cost Impacts of Jointness: Insights From the NPOESS Program*** Morgan Dwyer, Massachusetts Institute of Technology Zoe Szajnfarber, George Washington University Bruce Cameron, Massachusetts Institute of Technology Markus Bradford, Massachusetts Institute of Technology Ed Crawley, Massachusetts Institute of Technology ***Acquisition Risks in a World of Joint Capabilities: A Study of Interdependency Complexity*** Mary Maureen Brown, University of North Carolina Charlotte

The Decline and Fall of Joint Acquisition Programs

Andrew P. Moore—is a lead researcher in the CERT Insider Threat Center and a senior member of the technical staff at Carnegie Mellon University's Software Engineering Institute. He has over 20 years of experience developing and applying mission-critical system analysis methods and tools. Moore has worked for the Naval Research Laboratory (NRL) investigating high-assurance system development methods and has co-authored a book, two book chapters, and a wide variety of technical journal and conference papers. Moore received an MA in computer science from Duke University, a BA in mathematics from the College of Wooster, and a graduate certificate in system dynamics from Worcester Polytechnic Institute. [apm@sei.cmu.edu]

William E. Novak—is a senior member of the engineering staff at the Carnegie Mellon University Software Engineering Institute. He is a researcher, consultant, and instructor in the acquisition and development of software-intensive systems. Novak has over 30 years of experience with government acquisition, real-time embedded software and electronics product development, and business management. Novak has held positions with GE Corporate Research and Development, GE Aerospace, Texas Instruments, Tartan Laboratories, and GTE Automatic Electric Laboratories. Novak received his MS in computer engineering from Rensselaer Polytechnic Institute and BS in computer science from the University of Illinois at Urbana-Champaign. [wen@sei.cmu.edu]

Matthew L. Collins—is a current graduate student at the H. John Heinz III College at Carnegie Mellon University and a graduate assistant in the CERT Insider Threat Center. In addition to information security, Collins has focused his graduate studies on system dynamics, optimization, and public policy. Collins received his BS in business management from the McKenna School of Business at Saint Vincent College. [mlcollins@sei.cmu.edu]

Jay D. Marchetti—is a senior member of the technical staff at the Software Engineering Institute. Jay received his BS in EE, magna cum laude, from the University of Pittsburgh and his MSEE from the University of Rochester. Marchetti has worked in digital image processing at Eastman Kodak and in digital control systems at Contraves USA. He has architected and led the development of control system hardware, software, and systems for numerous real-time and embedded servo and communications products in the motion simulator, rail vehicle, and power distribution industries. Marchetti is the inventor on three U.S. patents, the most recent awarded in 2010. [jaym@sei.cmu.edu]

Julie B. Cohen—is a member of the Acquisition Support Program at the Software Engineering Institute (SEI), where she served for three years on the Transformational Communications System program. Prior to the SEI, Cohen was a program manager at both Brashear and Marconi. In the Air Force, Cohen worked in positions including the F-16 program office, the Flight Training program office, the Air Force Operational Test and Evaluation Center, and the Air Force Research Laboratory. She received her BS in EE from Carnegie Mellon University and an MS in EE from the Air Force Institute of Technology. She is a certified program manager professional and attained a Level 3 Certification as a DoD program manager. [jcohen@sei.cmu.edu]

Abstract

Studies have shown that joint acquisition programs are prone to experience larger cost and schedule overruns than single service programs, but that cost growth is not related to their generally larger sizes. This paper explains the unique and substantial cost growth of joint programs by describing an underlying causal mechanism that drives the observable schedule delays and cost overruns.

Through analysis of actual joint program performance data and the use of that data in a system dynamics model of acquisition program behavior, we characterize two primary sources of joint program cost growth: (1) requirements growth and rework due to a social dilemma that occurs for the Joint Program Office due to interactions among stakeholder programs, and (2) rework driven by more traditional causes such as contract underbidding and the resulting schedule pressure increasing defect levels, and then effectively depressing

productivity through the resultant rework. The combination of the two effects diminishes joint program performance significantly, explaining the previously identified degree of severity.

In joint programs, the slowing performance manifests itself as a cascade of departing stakeholder programs who are unwilling to accept the growing schedule and cost, ultimately resulting in the cancellation of the program.

Introduction

Joint Department of Defense (DoD) acquisition programs intend to provide a system, subsystem, or capability that will fulfill the needs of, and be funded or managed by, more than one DoD service or component. Joint programs are appealing because they offer at least two significant potential benefits: (1) reducing costs by developing one system as opposed to several differing ones, and (2) improving interoperability by providing a single system or capability that can be used for multiple purposes in multiple contexts.

Joint programs are noted for the unique challenges that they face organizationally (Lindsay, 2006), due in part to the tension between the individual programs and services needing to look out for their own interests, and the Goldwater-Nichols Act of 1986 (Goldwater-Nichols, 1986) that stresses the importance of all service branches working together both effectively and efficiently. Because of this seeming paradox there is a fundamental social dilemma at the heart of every joint program—a social dilemma known as a “Tragedy of the Commons” in which the shared commons is the development resource of the joint program office and the contractor. Both the joint program and its stakeholder programs are collectively worse off if the stakeholder programs choose to exploit the development resource for their individual gain by insisting on having custom requirements developed[1].

This paper describes research conducted to validate the nature of the joint acquisition program social dilemma and provide insight into mitigations of that problem. Through analysis of historical joint program performance data and the use of that data in a system dynamics model of acquisition program behavior, we characterize two primary sources of joint program cost growth: 1) requirements growth and rework due to a social dilemma that occurs for the Joint Program Office (JPO) due to interactions among stakeholder programs, and 2) rework driven by more traditional causes such as contract underbidding and the resulting schedule pressure increasing defect levels, and then effectively depressing productivity through the resultant rework. The combination of the two effects diminishes joint program performance significantly, explaining the previously identified degree of severity.

The work described here extends the model presented in the 2013 NPS Acquisition Research Symposium proceedings (Moore, Novak, Cohen, Marchetti, & Collins, 2013). The preliminary model presented in 2013 describes how the social dilemma was found to be operating in joint acquisition programs. Evidence supporting the 2013 model was derived from

[1] It is important to note that a “Tragedy of the Commons” situation does not always occur in a joint program. It may be the case that strong leadership from the joint program manager, or a highly cooperative culture within the program, will prevent it from happening. However, given the fact that the incentives align to favor unilateral action by the stakeholder programs and their services, unless specific preventative steps are taken, avoiding this social dilemma is more likely to be the exception rather than the rule.

- targeted workshops with the decision-makers and developers associated with a particular acquisition program, and
- the SEI’s general understanding of the problem gained through the regular conduct of Independent Technical Assessments (ITAs) on specific programs to determine why they are experiencing difficulties.

This paper describes the results from a detailed analysis of operational data from the same joint acquisition program including lessons learned from that data analysis, a refinement of the development segment of the simulation model based on that analysis, and a broadening of our data collection efforts necessary due to the limitations of the program performance data. We also present options for mitigating the joint program dilemma based on the modeling and analysis to date.

Program Data Collection and Analysis

We began our data collection by identifying data that was both captured by joint programs and relevant to our system dynamics model. Our initial data collection came from workshops with experts in the acquisition community. These workshops provided valuable qualitative insights that guided the construction of the system dynamics model. The experts were able to identify relationships between variables[2] in the model and to show general relationships between changes in variables. While the workshops identified relationships between variables, additional data was required to identify the magnitude of a change in a variable that had been caused by a change in another variable.

In order to obtain the quantitative data needed to improve our model, we looked at documents from past joint programs. In addition to the qualitative insights we received from the group modeling workshops, attendees also provided us with documents from past joint programs. These documents included code sizing information, staffing rates, and earned value metrics. In order to integrate this data into our model, we used a script to aggregate values from files representing different points in time into a single file.

Though we were able to collect data that supported our model, we were limited by the types of data that were recorded during joint programs, and further limited by the data we were both able and allowed to access. Although this is an expected limitation of the data collection process, it highlights an important aspect of our research: the social dilemmas inherent in joint programs are not well documented, nor are there metrics that can be universally applied to joint programs to determine a program’s “performance” or to predict a program’s outcome. This has led to the development of variables in the system dynamics model that reflect key social dynamics that we and other joint program professionals have witnessed in joint programs. We believe that these social dynamics are as important to a program’s success, if not more so, than the dynamics that play out in traditional software development. Furthermore, these underlying social dynamics may be a significant contributing cause of poor program performance that is reflected in traditionally recorded program data.

[2] See Appendix A for more information on system dynamics modeling and definitions of relevant terms.

Broadened Scope

The limitations of acquisition program data that is available require us to broaden our data collection efforts beyond that which is typically collected, recorded, or consciously observed by the Program Management Office (PMO). Other sources of data include expert opinion as collected through directed workshops, industry averages, and game-based experiments. Figure 1 shows a high-level overview of our process for our broadened data collection and analysis.

There are several explanatory theories in the literature related to the dynamics that were discussed in the group modeling workshops including cooperation/negotiation theory (Axelrod, 1997; Darling & Mumpower, 1990; Axelrod, 1984), social dilemmas (Kollock, 1998), the "Tragedy of the Commons" (Cross & Guyer, 1980; Hardin, 1968), and altruistic punishment (Fehr & Gachter, 2002). These theories provide possible explanations and potential causes of behavior observed in past joint programs. The research hypotheses described in the next section were derived based on these theories and our experience evaluating challenged programs. These hypotheses drive the collection of data to find confirming or refuting evidence, and the development of a simulation problem model that exhibits the problematic behaviors embodied by the data collected.

Evidence that runs counter to the subject hypotheses may require refinement of the hypotheses or testing of alternate hypotheses in subsequent work. Once the problem is accurately understood and modeled, the model structure and execution can help to understand the benefits associated with various mitigations to the problem. This may require refinement of the model into a more comprehensive simulation solution model that embodies approaches ranging from mandates to economic incentives as described in the literature. Any improved understanding and research results can be fed back into the explanatory theory of the problem. The simulation solution model can also serve as a basis for training or on-the-job decision support for joint program managers.

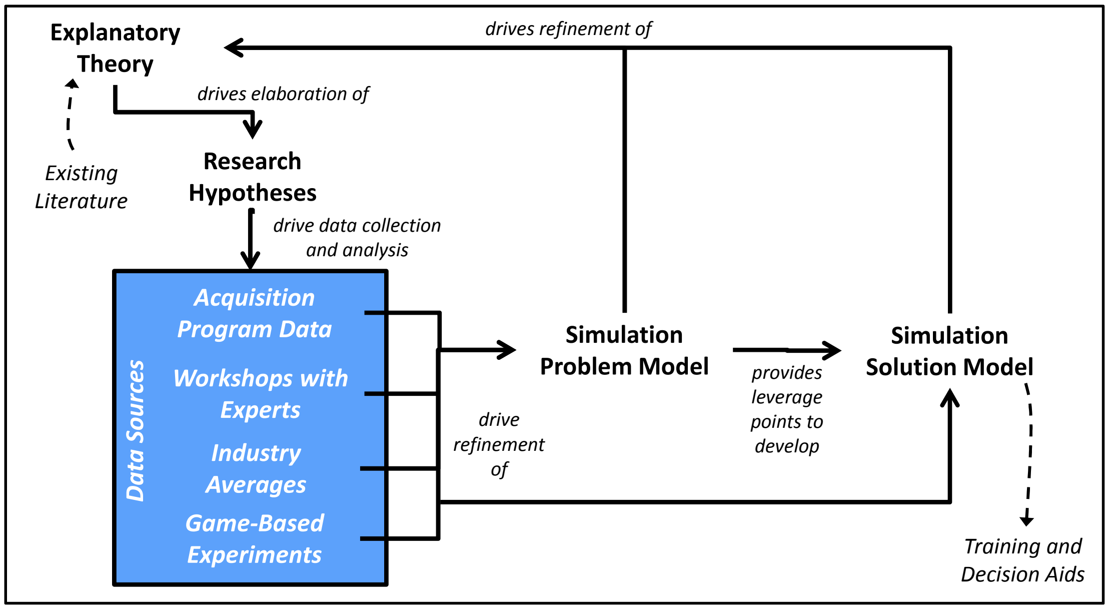


Figure 1. Simulation Modeling as Theory Building

Tracking Evidence

The task of tracking evidence regarding our research hypotheses becomes more complex as we broaden the scope of our data collection and analysis activities. We used a technique called assurance cases that helps document evidence that a software system satisfies its non-functional requirements.[3] In our case we are documenting the research hypothesis validation case.

Figure 2 provides an excerpt of the validation case tracking the evidence for research hypothesis 3. We use a subset of the regular assurance case notation: squared rectangles represent claims made, circles represent evidence provided that support (or refute) claims, and rounded rectangles represent context for the argument being made. Shapes with a triangle at the bottom indicate refinements that are yet to be made. In this paper these aspects of the validation case were omitted due to space limitations.

The high-level proposition is that joint acquisition programs exhibit the behavior of a “Tragedy of the Commons” social dilemma. This high-level proposition is comprised of the following three main hypotheses, of which only Hypothesis 2 is refined in the figure:

1. *Hypothesis 1*—Stakeholder programs request custom requirements after the baseline requirements are established, many of which the JPO accepts.
2. *Hypothesis 2*—Introducing additional requirements after the baseline requirements are established decreases the overall developer productivity during system development.
3. *Hypothesis 3*—Most stakeholder programs leave the joint program because they do not get their custom requirements accepted by the JPO or, if they are accepted, the development schedule, cost of implementing them, or the resulting quality does not meet their needs.

These three hypotheses form the basis of the proposed Tragedy of the Commons. The common resource being exploited is the JPO’s resources to develop a joint system. The custom requirements accepted after the baseline is established (Hypothesis 1) sets up a situation where developer productivity is lessened (Hypothesis 2). The JPO may feel that they have little option but to accept the custom requirements since they need to keep the stakeholder programs sufficiently satisfied that they do not leave the joint program (Hypothesis 3). However, if they continue to accept the additional custom requirements, overall developer productivity will be diminished to the point that cost and schedule will be adversely impacted. If stakeholder programs are overly demanding, this will ultimately lead to stakeholder programs leaving due to poor program execution, as indicated in Hypothesis 3. The joint program thus ultimately collapses when the stakeholder programs have little motivation to constrain their demand for custom requirements.

The validation case helps track the evidence concerning the truth or falsity of the hypotheses based on the current progress of the data collection, modeling, and analysis. Figure 2 shows only the refinement of Hypothesis 2 (due to space limitations). In particular, evidence is documented that supports the case that the model accurately reflects the behavior of the acquisition program in terms of the diminishing returns on developer productivity (Model Test Evidence). Note that the model validity requires both that the model

[3] See http://www.sei.cmu.edu/dependability/tools/assurancecase.

exhibit the *right behavior* of the joint acquisition program (Model Validity Evidence 2.1-2.3), *and* that it must exhibit this behavior for the *right reasons* (Model Validity Evidence 2.4-2.5). As we have mentioned, sources of evidence include program data, subject matter expert workshops, industry data, and game-based experiments. Hypothesis 3 is the subject of a game-based experiment that will be outlined in the section titled Game-Based Experimentation.

Alternate Hypotheses

The discussion so far has focused on joint program problems arising from the late introduction of custom requirements from stakeholder programs. This may not be the only reason, or even the most prominent reason why joint programs fail. We continue to look for evidence supporting or refuting this hypothesis. In addition, we have two alternate hypotheses for the failure of joint acquisition programs: Underbidding the Contract and Doing the Easier Work First:

> *Underbidding the Contract*: When contractors bid a lower price or a shorter duration in order to win a contract, they may make assumptions that all will go well on the program, and that there will be no setbacks. In reality, setbacks are inherent in the nature of large-scale development programs and, due to the complexity of joint programs, one setback may have serious cascading effects on schedule and cost. Underbidding the contract can lead to schedule pressure and shortcuts in quality processes that in turn lead to increased rework at a later date, increases in firefighting, and staff burnout.
>
> *Doing the Easier Work First*: Developers have a strong motivation to show good progress early in the development effort. The desire to increase stakeholder support and buy-in fuels the motivation of the developer to show substantial progress early in the program. This method of development, however, can lead to the *bow wave effect* (Novak & Levine, 2010), which occurs when a developer puts off the most difficult tasks and focuses primarily on “quick wins” in order to show good progress. This results in the early completion of lower risk, easier requirements. This naturally delays the development of larger, higher-risk requirements until later in the project. Later development of more difficult requirements has been shown to cause additional schedule slips and increased costs.

In reality, joint programs fail for multiple reasons. Our analysis in the rest of the paper investigates a particular combination of factors that may lead to program failure.

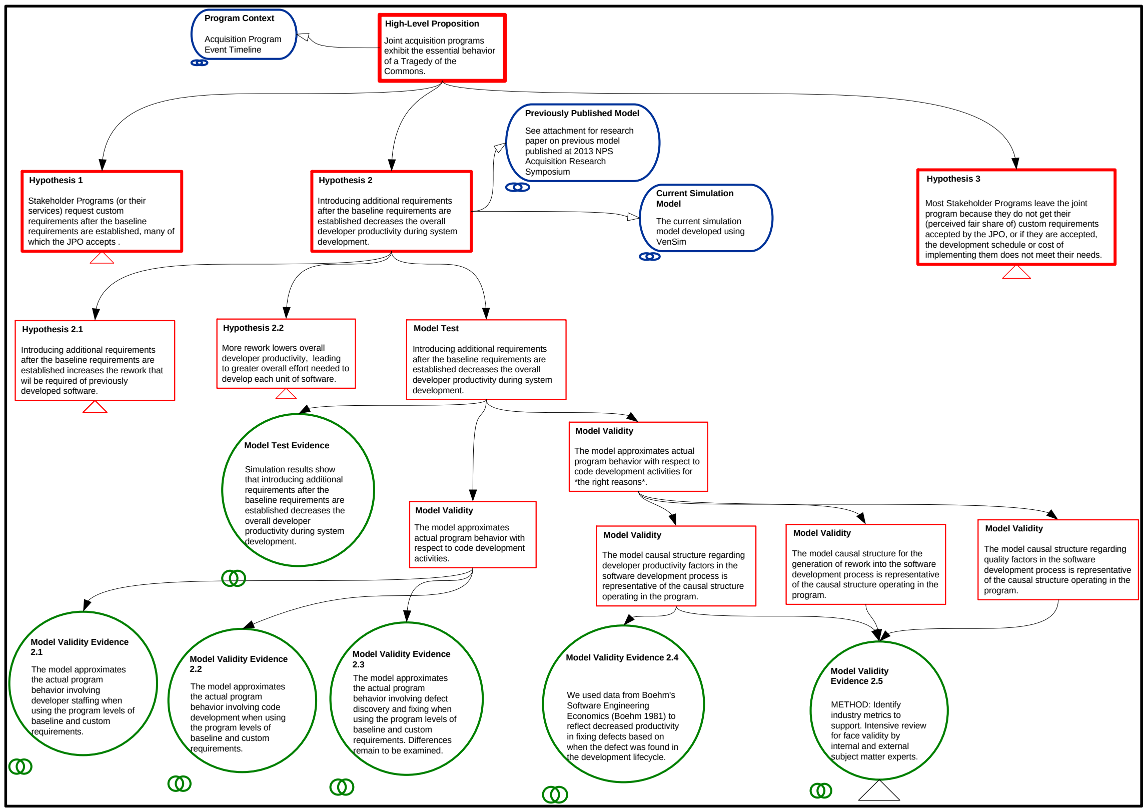

Program Context
Acquisition Program Event Timeline
High-Level Proposition
Joint acquisition programs exhibit the essential behavior of a Tragedy of the Commons.
Previously Published Model
See attachment for research paper on previous model published at 2013 NPS Acquisition Research Symposium
Current Simulation Model
The current simulation model developed using VenSim
Hypothesis 1
Stakeholder Programs (or their services) request custom requirements after the baseline requirements are established, many of which the JPO accepts .
Hypothesis 2
Introducing additional requirements after the baseline requirements are established decreases the overall developer productivity during system development.
Hypothesis 3
Most Stakeholder Programs leave the joint program because they do not get their (perceived fair share of) custom requirements accepted by the JPO, or if they are accepted, the development schedule or cost of implementing them does not meet their needs.
Hypothesis 2.1
Introducing additional requirements after the baseline requirements are established increases the rework that wil be required of previously developed software.
Hypothesis 2.2
More rework lowers overall developer productivity, leading to greater overall effort needed to develop each unit of software.
Model Test
Introducing additional requirements after the baseline requirements are established decreases the overall developer productivity during system development.
Model Test Evidence
Simulation results show that introducing additional requirements after the baseline requirements are established decreases the overall developer productivity during system development.
Model Validity
The model approximates actual program behavior with respect to code development activities for *the right reasons*.
Model Validity
The model approximates actual program behavior with respect to code development activities.
Model Validity
The model causal structure regarding developer productivity factors in the software development process is representative of the causal structure operating in the program.
Model Validity
The model causal structure for the generation of rework into the software development process is representative of the causal structure operating in the program.
Model Validity
The model causal structure regarding quality factors in the software development process is representative of the causal structure operating in the program.
Model Validity Evidence 2.1
The model approximates the actual program behavior involving developer staffing when using the program levels of baseline and custom requirements.
Model Validity Evidence 2.2
The model approximates the actual program behavior involving code development when using the program levels of baseline and custom requirements.
Model Validity Evidence 2.3
The model approximates the actual program behavior involving defect discovery and fixing when using the program levels of baseline and custom requirements. Differences remain to be examined.
Model Validity Evidence 2.4
We used data from Boehm's Software Engineering Economics (Boehm 1981) to reflect decreased productivity in fixing defects based on when the defect was found in the development lifecycle.
Model Validity Evidence 2.5
METHOD: Identify industry metrics to support. Intensive review for face validity by internal and external subject matter experts.

Figure 2. Excerpt From Validation Case for Research Hypothesis 2

Simulation Problem Model

This section extends and refines the Developer Segment of the model presented at the 2013 NPS Acquisition Research Symposium (Moore, Novak, Cohen, Marchetti, & Collins, 2013). Readers may find familiarity with that paper useful as a context for the work reported here, but it is not necessary in order to understand the progress we have made since then, as reported in this paper. We first describe the Development and Rework segment of the model developed based on the program data available, and the correspondence of the calibrated model to the acquisition program performance.

Development and Rework Model Segment

Figure 3 shows the basic stock and flow structure of the portion of the model that is focused on software/system development activities. The stock of Development Work Remaining starts out with the full scope of the software artifacts to be developed according to the initial baseline requirements. As new custom requirements are generated, it adds to the work to be done. Once developed, the artifacts that need to be tested accumulate in the Test Work Remaining stock. Artifacts either pass their tests and go to Work Completed, or fail their tests and go to Failed Work Remaining. Upon fixing the artifacts they must pass their conformance tests before being released. Work released that needs to undergo rework, possibly based on the introduction of new requirements, accumulates in the Rework Remaining stock. Reworked artifacts must pass their conformance tests as well.

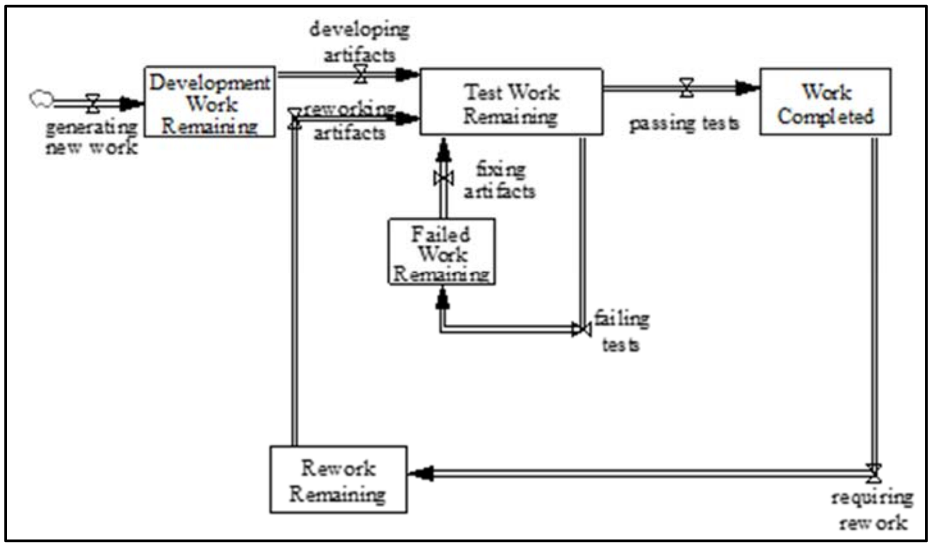


Figure 3. Development and Rework Stocks and Flows

Figure 4 extends the stock and flow structure shown in Figure 3 and provides connection points with the overall model shown in Appendix B. We focus here on the productivity factors that regulate the flow of artifacts between stocks. The rate of accepting custom requirements influences the generation of new work, as shown. Three productivity factors are represented: development, rework, and test. Average measures are calculated based on the development and test staff available. Development staff are split between initial development work and rework. Productivity measures start at a normal value, but are adjusted based on other factors that will be shown later in this section.

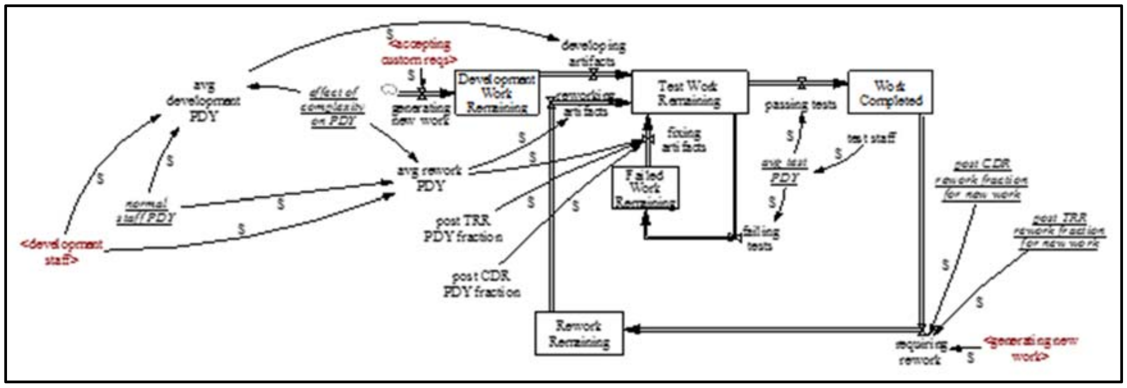


Figure 4. Productivity Factors

As shown in the lower right of the figure, rework of previously completed work is generated based on the rate at which new custom requirements generate new work to be done. The new requirements were not previously considered, and so have the general effect of undermining previously completed work. This effect becomes worse the later that the new requirements are introduced into the development process. Our model assumes three stages of rework generation. Up to the point of the Critical Design Review (CDR), no rework is generated. After CDR, but before the Test Readiness Review (TRR), the amount of rework generated is a fraction of the generating new work rate given by the variable post CDR rework fraction for new work. Any new requirements introduced after TRR are given by the product of the generating new work rate and the variable post TRR rework fraction for new work. Similarly, post CDR PDY fraction and post TRR PDY fraction describe reductions in defect repair productivity based on when the defect was found in the development lifecycle, as indicated in past research (Boehm, 1981).

Figure 5 refines the development and rework model segment to the next level showing the negative effects of complexity on productivity and defect introduction (shown in blue) and the additional reinforcing negative effects off schedule pressure on defect introduction (shown in red). Complexity is measured relative to the baseline, i.e., as a ratio of the total work (baseline development work + total custom work) and the baseline development work, as illustrated in the upper left corner of the figure. The greater the relative complexity, the lower the productivity and the higher the defect rates are for artifacts developed. There are two *effect functions*[4] that determine the impact of complexity on productivity and defect injection: effect of complexity on PDY and effect of complexity on quality issues, respectively. This is an admittedly simple view of how complexity could impact software and system development. We continue to evaluate whether it is adequate for the purposes of our modeling efforts.

Another significant factor that affects defect injection is schedule pressure. Figure 5 shows a reinforcing feedback loop in red named Pressure-Induced Defects. As shown in the bottom middle of the figure, the fraction of artifacts requiring change determines the failing or passing of software tests. As the fraction increases, more test work remains, and the Perceived Completion Date may need to be extended. In accordance with the extent to which the perception of time needed does not match the time officially available to complete the development, schedule pressure rises. Greater schedule pressure then leads to even higher defect rates. The hidden assumption here is that overall schedule pressure puts pressure on the software developers, who then try to develop the software faster, and as a result are less careful with their work, and may even deliberately reduce or omit altogether some quality assurance processes.

[4] An *effect function* is a device used in system dynamics modeling that explicitly describes the mathematical relationship between two specific model variables over time.

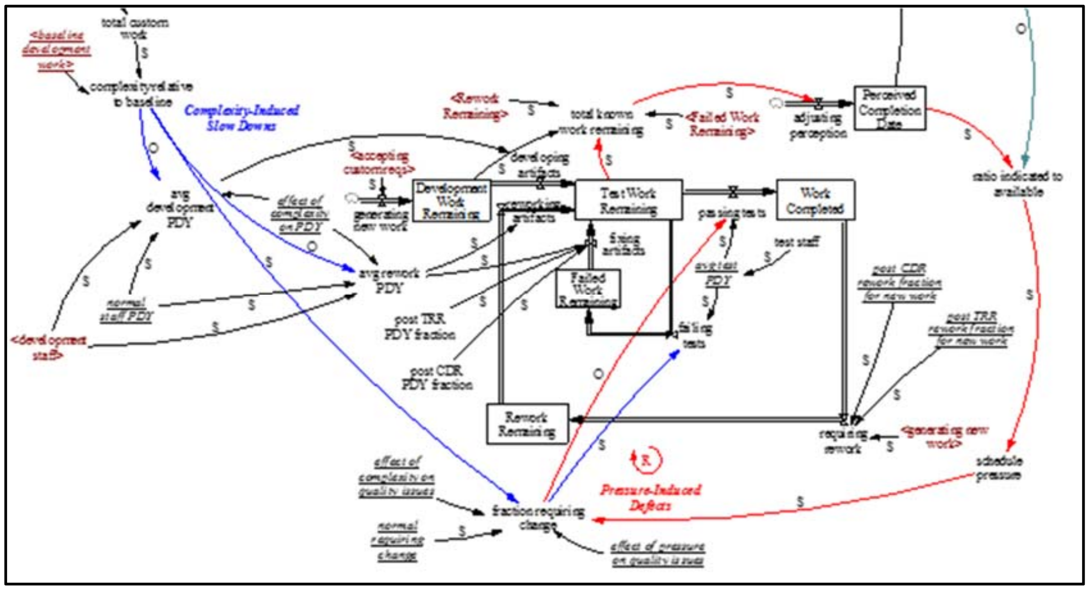


Figure 5. Development and Rework Model Segment

Model Correspondence With Program Data

Figure 6 shows the level of Work Completed as determined from model execution and as compared with the actual performance of the acquisition program with which we collaborated. Certain irregularities in the data were smoothed out through discussion with our program collaborators. In addition, the model execution is based on the specific values used to instantiate the variables discussed in the development and rework model segment discussed in the last section. While these underlying variables are not explicitly known in the case of the acquisition program, we have derived values which, when used in the simulation, allow a relatively close correspondence with other (known) program performance variables.

The correspondence of program and model work completed shown is generated using the program requirements introduction shown in Figure 7a. The baseline requirements are those generated up to PDR at month 8. Any new requirements introduced after PDR are assumed to be custom requirements demanded by stakeholder programs. Figure 7b shows the level of development staff over time for the acquisition program and for the model simulation. In this case, we did not use the program data as input to the model, but instead developed a staffing segment of the model to reflect actual staffing. This allows us to regulate staffing for testing the model under different operational conditions in the next section.

Development and rework productivity variables are based on a normal average developer productivity of 100 source lines of code (SLOC) per person-month. This normal value is adjusted based on the complexity relative to baseline variable and the effect of complexity on PDY effect function shown in Appendix C. Similarly, the fraction of artifacts requiring change variable is based on a normal value of 1/5th. This normal value is again adjusted based on the complexity relative to baseline variable, but here the effect of complexity on quality issues effect function (as shown in Appendix C) is used. The variable schedule pressure also modifies the fraction requiring change based on the effect of pressure on quality issues effect function. These two factors have a multiplicative effect on the quality issues.

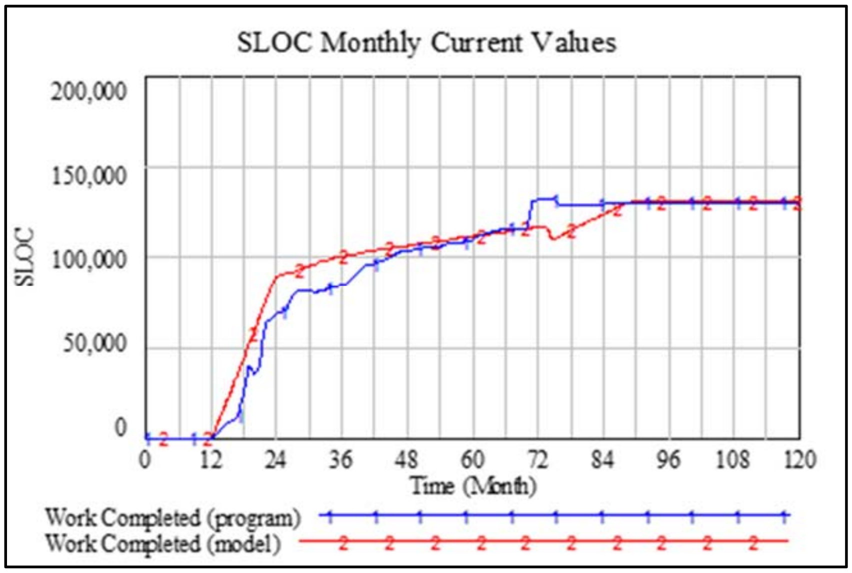


Figure 6. Acquisition Program vs. Model Cumulative Work Completed[5]

Rework generation is based on the two multipliers of the generating new work variable:

post CDR rework fraction for new work = 0.6

post TRR rework fraction for new work = 0.9

CDR occurs at month 12 and TRR occurs at month 28. As an example, after CDR but before TRR for a period of 16 months, the rate of requiring rework is 0.6 of that of generating new work. From month 28 until project completion at about month 90, the rate changes to 0.9. Before month 12, no rework is generated.

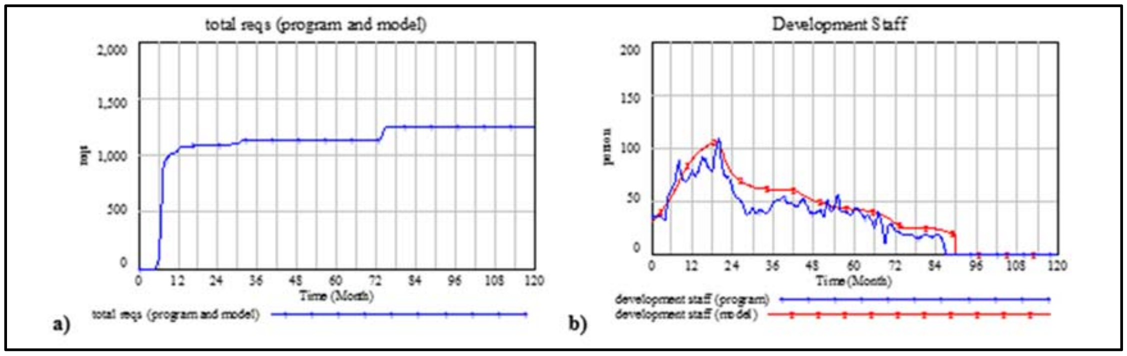


Figure 7. Assumed Values for a) the Program Requirements Introduction; b) Development Staffing Levels

[5] This and subsequent graphs were generated using the Vensim® modeling tool. These are all behavior-over-time graphs and, as such, the X-axis for these graphs is specified in months (120 months—10 years—is the duration of this simulation). Each simulation run is specified as individual graphs distinguished with a number label (1 and 2 in **Error! Reference source not found.**) as specified in the legend below the graph.

The correspondence shown in Figure 6 is clearly not perfect. The initial slope of the accumulated developed code is about the same for both the model and the program through month 24, and again from month 24 to the software development completion at approximately month 90. The change in slope for the model, and presumably for the actual program as well, is due to a shift from artifact development to artifact test and rework. At month 24 the model rises to a higher level before the inflection point than indicated in the program data. In addition, there is a significant rise in additional requirements at month 74 (seen in Figure 7a), which leads to rapid development in the actual program data, but the same rate of increase in the model. One explanation for this latter effect is that there was a surge of developer overtime and effort in the last months in the acquisition program that is not reflected in the model. We will continue to try to resolve these differences through discussions with our collaborator.

Even if the above model correspondence were more precisely aligned, this would not in and of itself indicate that the model is correct. Other, different models could generate essentially this same behavior. More points of conformance are necessary to gain greater confidence that our model is sufficiently accurate to predict program response to alternate mitigation approaches. While it is useful to explore the significance of the simulation outputs at this intermediate stage, we are working to further analyze program data particularly in the areas of defect/fix rates and rework.

Preliminary Observations

Hypothesis 3 in the validation case described previously states that the more time that passes between the establishment of baseline requirements and the introduction of additional requirements, the lower the overall developer productivity will be during system development. We measure developer productivity, i.e., KSLOC developed per person-month of effort, over the lifetime of the development effort. We normalize developer productivity by dividing actual developer productivity by the developer productivity seen in the model for increasing levels of custom requirements introduced after the baseline. As shown in Figure 8, since the baseline requirements are those established before PDR, custom requirements are those introduced after PDR. We test our simulation by increasing the levels of these requirements.

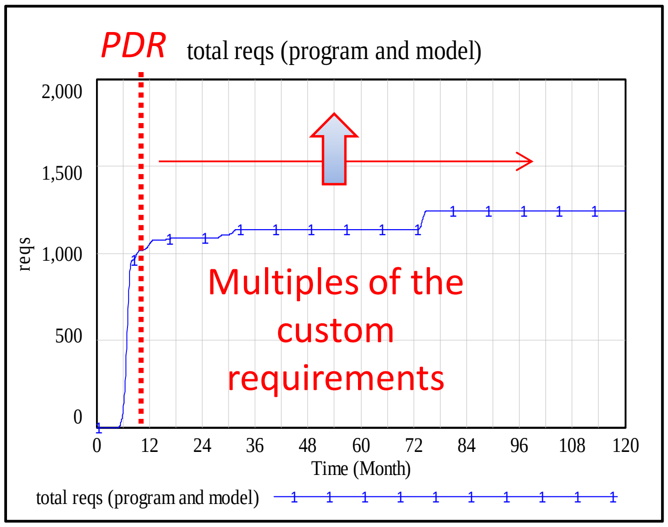


Figure 8. Increasing Levels of Custom Requirements

Figure 9 shows the results of executing the model for increasing levels of custom requirements, where the custom requirements introduced after PDR are measured as a percentage of the total program requirements. The three-dimensional surface in the graph simultaneously shows how developer productivity varies as the realism of schedule setting varies from low to high. The program that we considered had just over 20% of the custom requirements introduced after PDR and was assessed to have a medium level of schedule realism. The developer productivity for the actual program was used as the baseline for normalizing developer productivity and therefore intersects at a productivity level 1.

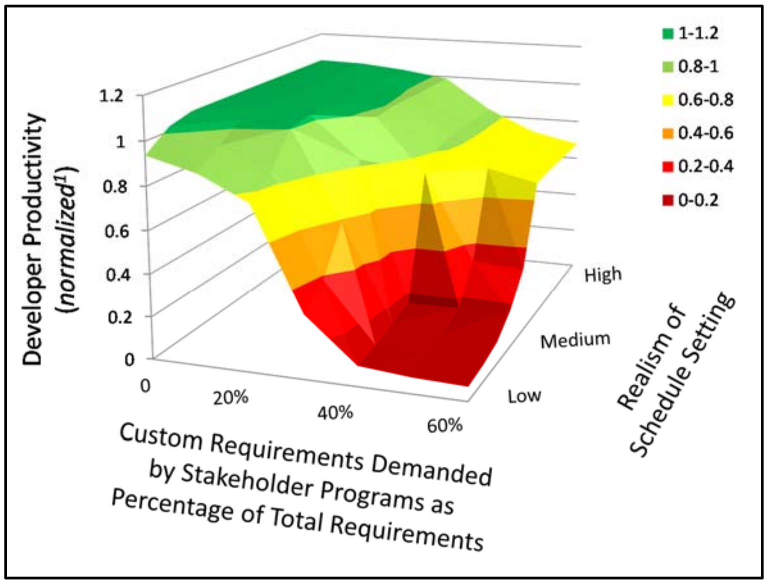


Figure 9. Tipping Point of Developer Productivity

The precipitous drop in developer productivity as schedule realism declines and the level of custom requirements increase is consistent with our Hypothesis 3, since multiplying the actual program custom requirements has the effect of increasing the requirements introduced after the baseline requirements are established. The requirements growth increases the complexity of the system under development which decreases the developer productivity and increases the defect injection rate as noted in the model description in the section titled Simulation Problem Model. We included the realism of schedule setting as a variable in the simulation to test the impact on developer productivity of contract underbidding and the aggressive initial development schedule that results. Interestingly, as shown in Figure 9, schedule realism does not play a large role in diminishing developer productivity where the percentage of custom requirements is low, say below 10 or 20%. However, with larger percentages of custom requirements, lack of schedule realism dramatically exacerbates the drop of developer productivity. The combination of a high level of custom requirements and lack of schedule realism (again, potentially due to initial contract underbidding) is the most deadly. Note also that high levels of schedule realism can buffer programs from the most severe drops of developer productivity due to high levels of custom requirements.

Figure 10 shows that average developer productivity can be improved by decreasing custom requirements or increasing realism of the development schedule. While this may seem to be an intuitive conclusion, the graph suggests that there is dramatic improvement possible in these areas.

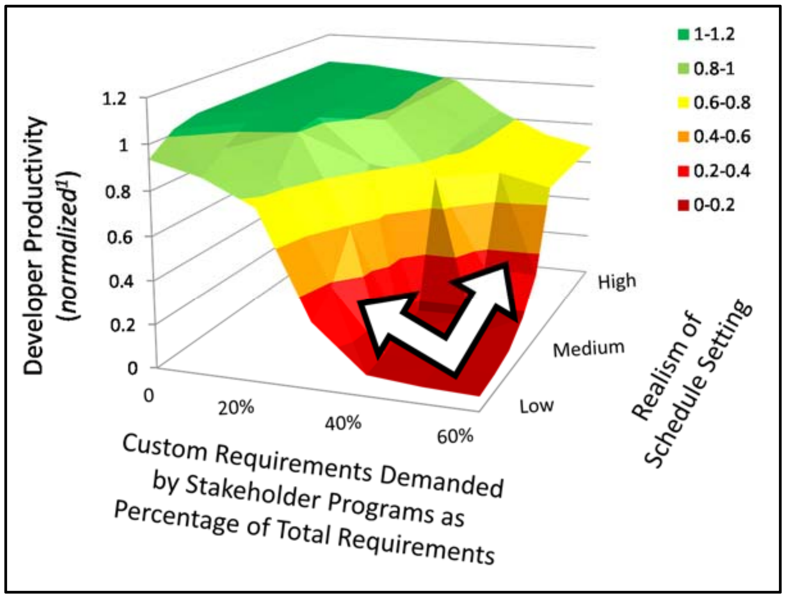


Figure 10. Directions for Improved Performance

Three potential mitigation strategies that address the basic goals of decreasing custom requirements and increasing schedule realism are: *Authority Mandate, Altruistic Punishment,* and *Incentivize Schedule Accuracy*. Authority Mandate refers to the use of an overarching authority that manages and enforces the sensible use of the shared resource, thus avoiding the risk of exploitation and overuse. Altruistic Punishment enables participants to punish perceived uncooperative (i.e., exploitative) participants through a mechanism such as a financial penalty. The Incentivize Schedule Accuracy approach provides incentives for accurate estimates so as to prevent the onset of significant schedule pressure that drives many of the subsequent problems leading to poor program performance and loss of stakeholder confidence. These approaches are elaborated in Table 1 with the primary benefits and some of the possible adverse side effects.

Table 1. Mitigation Areas Based on Simulation Behavior

Goal	Mitigation Strategies	Benefits	Possible Side Effects
Decrease Custom Requirements (CRs)	Authority Mandate	Joint Program Office has ability to deny CRs without risking defection	Forced participation increases Stakeholder Program (SP) animosity and SP collusion to sabotage the Joint Program
Decrease CRs	Altruistic Punishment	Limits CR growth by penalizing excessive CR demands (at cost to all)	Conflict may escalate and cause SPs to retaliate against each other
Increase Schedule Realism	Incentivize Schedule Accuracy	Limits likelihood of developer getting behind schedule	Taking shortcuts that reduce quality in order to achieve schedule

While there are many other potential mitigation strategies that can be applied to address this problem, these three are sufficient to give a sense of the range of possible options.

Game-Based Experimentation

The incorporation of real-world data into any model will benefit the model's ability to track actual historical behavior, as well as potentially predict future behavior, with increased fidelity. In our research program performance data acquired from several past joint programs was critical in calibrating certain aspects of the model. Specifically, cost, schedule, requirement additions, and KSLOC trends over the course of the programs were utilized to fine-tune effect functions within the model relating to development efficiency and rework. This utilization of real program data helped to provide higher correlation of the model to actual program performance data for a variety of outputs as shown, such as in Figures 6 and 7b.

However, due to the relative scarcity of actual joint program data, our dependence on program metrics data for calibrating the model was recognized early on as an issue that we needed to address. Furthermore, areas of the model other than cost, schedule, and development progress—for example the social dimensions such as the stakeholder's intent to remain engaged or, conversely, to defect from the joint program in favor of a "go it alone" approach—are not represented in the metrics normally collected by a joint program. There are good reasons for this. While professionals are expected to exhibit integrity in their professional actions and decisions, they are subject to the effects of incentives, especially when those incentives are strong, such as in matters of salary, attractive work opportunities, and promotions. As the workshops that were conducted with the JPOs and their development teams showed, not all of the behaviors exhibited by joint program stakeholders are socially admirable—and may not be willingly revealed. As a result, in order to gain insight about relevant feelings of fairness, cooperation, and confidence among stakeholders and the JPO we needed another vehicle to provide real-world data. Specifically, we wanted to answer the question: What is the likelihood of joint program stakeholder program defection as a function of:

- custom requirement acceptance rates?
- joint program schedule performance?
- perceived fairness of treatment?

The game-based experiment is intended to provide empirical data upon which we can test Hypothesis 3 described in the validation: Most Stakeholder Programs leave the joint program because they do not get their (perceived fair share of) custom requirements accepted by the JPO, or if they are accepted, the development schedule or cost of implementing them does not meet their needs.

Experiment Design

The game is intended to exemplify aspects of the "Tragedy of the Commons" nature of a joint acquisition program, where the lesson is that "Individually optimal decisions lead to collectively inferior solutions." It shows that, left unchecked, the collective custom feature demands of the stakeholder programs can require levels of effort and produce complexity and risk that can overwhelm the development capability of the JPO. It can also impact its ability to maintain its initial rate of development, causing the schedule to slip and miss the critical "need dates" of stakeholders, making them want to leave the program. Unfortunately, refusing to accept these demands can also cause stakeholders to become disenchanted and try to leave the program (in favor of developing a custom, "siloed" system), thus undermining the joint program viability. When enough stakeholder programs leave, eventually the program becomes irrelevant as a joint effort, or is cancelled outright.

The game is computer-based, with the human subject (hereinafter “the player”) interacting with the JPO and *N* other joint program stakeholders, where *N*, an experimenter setting, will normally be in the range [2, 5]. Figure 11 shows the game interface for N = 3 stakeholder programs. The JPO and all other stakeholders may be other players in the game, or computer-implemented automata. The game is hosted on a web server to accommodate players from any geographic locale; all that is required to play is a web browser and an Internet connection.

The game proceeds through rounds of play, where each stakeholder, having a set of custom requirements that they want to have accepted for implementation as part of the joint program, requests some number of them to be accepted by the JPO. The JPO and automaton stakeholder behaviors are deterministic, though aspects of their behavior are tunable via experimenter settings. In each round, the JPO may accept or reject the player’s custom requirements requests, as it will for the other stakeholder requests. As the game proceeds the acceptance and rejection of requests by all stakeholders is displayed via bar charts, permitting the human subject to assess their perceived fairness of the JPO’s decisions. As more and more custom requirements are accepted by the JPO, the expected delivery schedule, presented with some degree of uncertainty on a timeline, will (as in an actual joint program) generally slip. To simplify the game, the degree of this schedule slip is the sole measure of program “performance” for the player (i.e., cost growth is not considered in the game, but the viability of the estimated delivery date may be determined by comparing it to the player’s assigned “need date” for the system).

The schedule slip and the JPO’s unpredictability present a tension for the human subject in that the game instructions make clear that their score can be maximized by getting as many of their custom requirements accepted as possible, while also having the system deliver within their need date. As the JPO accepts more custom requirements, the risk of missing essential deadlines escalates. During each round of play the human subject has two options: request the number of custom requirements they deem appropriate, or “defect” from (i.e., leave) the joint program. If they elect to defect, the player must specify the reason as either “too many requests denied” or “schedule slips.” Data on all player, stakeholder, and JPO actions, along with the experimenter settings, and post-game questionnaire answers, are recorded for each game. Subsequent reports on aggregated data will be used to assess data trends so that they may be utilized within the joint program system dynamics model.

While no experimental design can guarantee the replication of “real-world” behavior, by conducting the experiments with actual acquisition program staff, and designing the experiment as a competitive game with a financial incentive to win, this experiment attempts to reproduce as many of the key aspects of the joint acquisition context as possible.

Summary and Conclusions

This paper presents research that is being conducted to investigate whether a social dilemma, known as a “Tragedy of the Commons,” is the cause of the observable schedule delays and cost overruns that are common to joint programs. Three main hypotheses are described that form the basis of the proposed social dilemma, as well as two alternative hypotheses that describe other potential causes of joint program failure. We test these hypotheses through data analysis, subject matter expert workshops, linkage to past research, and game-based experiments.

The focus of the paper is the presentation of a simulation model as the embodiment of our current understanding of the problem, and as a means to test potential mitigations. The model described extends the model presented in the 2013 NPS Acquisition Research Symposium proceedings (Moore, Novak, Cohen, Marchetti, & Collins, 2013). The 2013 model is updated to reflect our analysis of quantitative data from an actual joint acquisition program. The model includes the rework cycle and the relevant productivity and defect injection factors that influence software development rework in joint programs, and contribute to the social dilemma with which joint program managers are faced. In the section titled Model Correspondence with Program Data, we compare the results from our model to the actual program data.

While we need additional data to further validate the model, this correspondence between actual program data and our model shows the increasing realism of the model at this stage of development. Our preliminary observations from the model show a potential tipping point in developer productivity related to custom requirements demanded by stakeholders and realism of schedule setting. We provide initial strategies that aim to address increasing custom requirements and to improve schedule realism. After analyzing and modeling the initial data, we found a need for additional information that is not typically recorded in the course of joint programs. In the section titled Game-Based Experimentation, we describe the design of a game-based experiment to collect data related to stakeholder defection from joint programs.

There are many areas of potential future work in using these techniques to analyze acquisition program behavior. One area where new analysis will be needed is an increasingly common type of joint program that is developing a capability that is to be integrated into existing systems, rather than developing a new stand-alone system. The “capability” approach is becoming more prevalent as funding dictates that systems will have longer lifetimes in the field, and will thus require more upgrades to remain current. There are important differences between developing *capabilities* and developing *systems*, focusing on the fact that capabilities must be integrated into existing systems, and may not be able to be extracted from those systems once they’ve been integrated—increasing the level of risk to the receiving system. This in turn undermines the level of trust that the receiving system program may have in the joint program, exacerbating the social dilemma that is already at work, and which requires trust to be overcome. This trend is becoming significant because of the natural role that software plays in providing upgraded capabilities in fielded systems—and thus understanding how to more successfully develop and deploy those software capabilities will be essential.

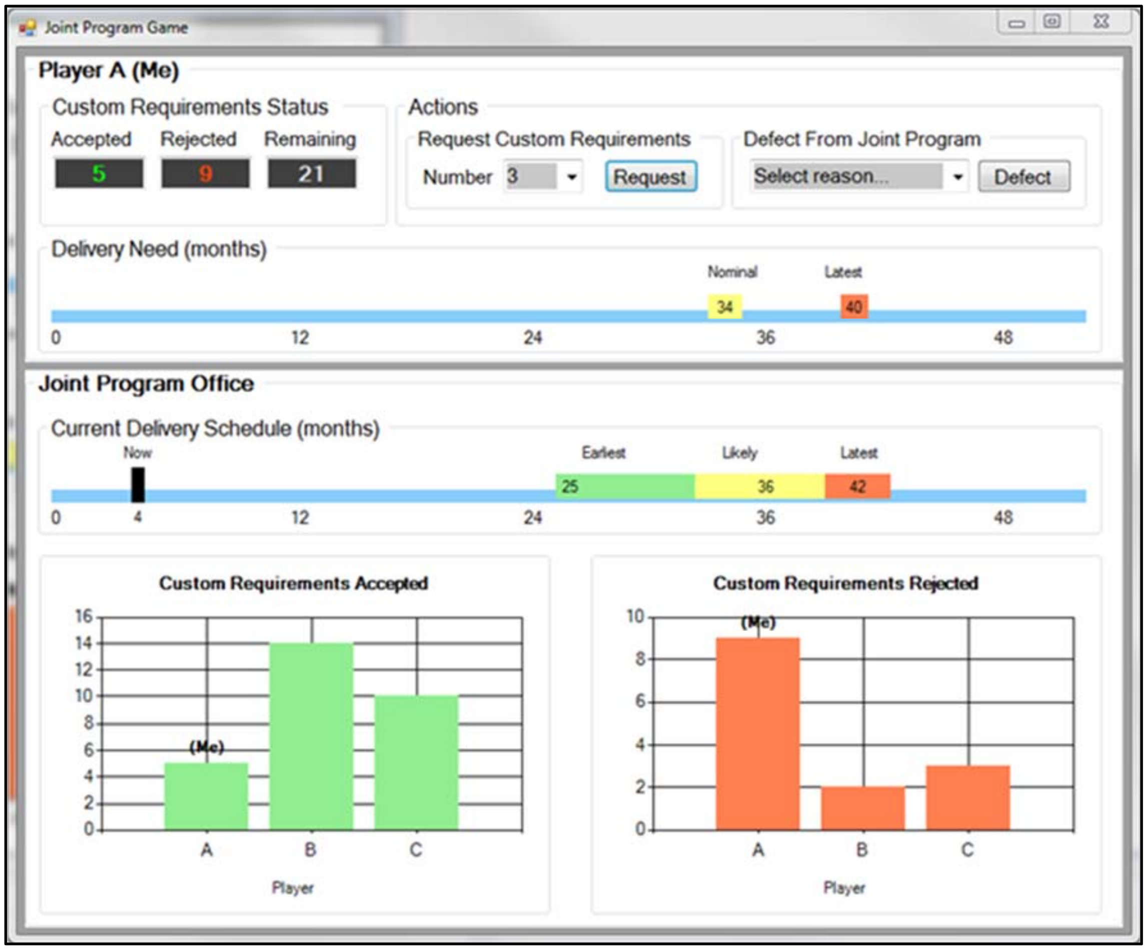


Figure 11. Prototypical Experimental Game Screen for 3 Stakeholders (*N* = 2)

Another area of potential future work is the development of educational tools based on the simulation model that would help joint program professionals recognize the social dilemma that is almost ubiquitous in joint programs. Training based on these results would provide insight into the dynamic organizational behavior present in joint programs, and better prepare acquisition professionals to manage joint programs. Looking beyond traditional classroom education, the system dynamics model also offers the opportunity to create a “management flight simulator” for acquisition that can provide experiential learning from a hands-on simulation to give acquisition staff a deeper and more intuitive understanding of these acquisition dynamics. These are important avenues to pursue given the challenges of both quantity and experience that continue to face the acquisition workforce.

Finally, one of the most advanced applications of the technology would be to forecast likely future acquisition program performance for different “what if?” scenarios using a parameterized system dynamics model that can be adapted to emulate specific programs. The development of a management decision support tool would require a high fidelity model of software acquisition, and the intensive application of detailed measurement data from software development efforts in order to provide sufficient predictive value to be of value. This type of capability would allow for the comparison and analysis of the likely outcomes of different decision alternatives when a critical decision must be made. It could also be used to predict the likely impacts (as well as the potential unanticipated side-effects) of new or modified policies on the performance and outcomes of the acquisition programs they are intended to regulate.

We see significant potential for the application of this technology to improve the effectiveness of the acquisition system from the level of individual acquisition staff members, to program managers, to that of acquisition executives and policy makers. It is only through a detailed analysis and understanding of the inner workings of the mechanisms of acquisition that we will be able to make real progress in making them operate more efficiently.

References

Axelrod, R. (1984). *The Evolution of Cooperation*. New York: Basic Books, Inc.

Axelrod, R. (1997). *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration.* Princeton: Princeton University Press.

Boehm, B. (1981). *Software Engineering Economics*. Saddle River: Prentice Hall.

Cross, J. G., & Guyer, M. J. (1980). *Social Traps*. Ann Arbor: University of Michigan Press.

Darling, T. A., & Mumpower, J. L. (1990). Modeling Cognitive Influences on the Dynamics of Negotiations. In R. H. Sprague (Ed.), *Proceedings of the 23rd Hawaii International Conference on System Sciences. IV*, pp. 22-33. Washington, DC: IEEE Computer Society Press.

Darling, T. A., & Richardson, G. P. (1990). A Behavioral Simulation Model of Single and Iterative Negotiations. *Proceedings of the 1990 International System Dynamics Conference* (pp. 228-241). Chestnut Hill: System Dynamics Society.

de Jong, S., & Tuyls, K. (2011). Human-Inspired Computational Fairness. *Autonomous Agent Multi-Agent Systems*, 103-126.

Fehr, E., & Gachter, S. (2002, January). Altruistic Punishment in Humans. *Nature*, 415, 137-140.

Goldwater-Nichols. (1986). *Department of Defense Reorganization* Act of 1986, PL 99-433. United States Statutes.

Hardin, G. (1968, December). Tragedy of the Commons. *Science*, 162, 1243-1248.

Kollock, P. (1998, August). Social Dilemmas: The Anatomy of Cooperation. *Annual Review of Sociology*, 24, 183-214.

Lindsay, J. R. (2006). War Upon the Map: *The Politics of Military User Innovation*. Political Science. Cambridge: Massachusetts Institute of Technology.

Meadows, D. (2008). *Thinking in Systems: A Primer*. White River Junction, VT: Chelsea Green Publishing.

Moore, A. P., Novak, W. E., Cohen, J. B., Marchetti, J. D., & Collins, M. L. (2013). The Joint Program Dillemma: Analyzing the Pervasive Role that Social Dilemmas Play in Undermining Acquisition Success. *Proceedings of the NPS Acquisition Research Symposium*. Naval Postgraduate School.

Novak, W. E., & Levine, L. (2010). Success *in Acquisition: Using Archetypes to Beat the Odds.* Technical Report, Software Engineering Institute, Pittsburgh, PA. Retrieved from http://www.sei.cmu.edu/library/abstracts/reports/10tr016.cfm

Sterman, J. D. (2000). *Business Dynamics: Systems Thinking and Modeling for a Complex World.* McGraw-Hill.

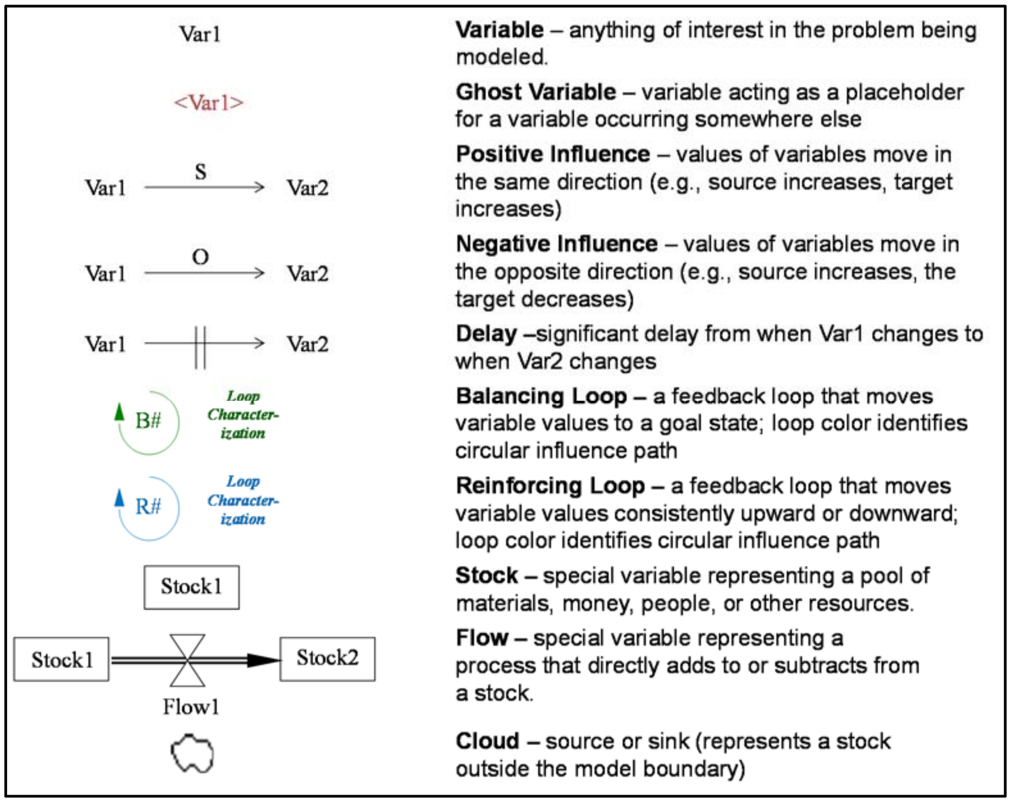
Appendix A: System Dynamics Background

The system dynamics method helps analysts model and analyze critical behavior as it evolves over time within complex socio-technical domains. A key tenet of this method is that the dynamic complexity of critical behavior can be captured by the underlying feedback structure of that behavior. The boundaries of a system dynamics model are drawn so that all of the enterprise elements necessary to generate and understand problematic behavior are contained within them. The method has a long history, as described in (Sterman, 2000) and (Meadows, 2008).

System dynamics and the related area of systems thinking encourage the inclusion of “soft” factors in the model such as policy, procedural, administrative, and cultural aspects. The exclusion of soft factors in other modeling techniques effectively treats their influence as negligible, which is often an inappropriate assumption. This holistic modeling perspective helps identify mitigations to problematic behaviors that are often overlooked by other approaches.

Figure 12 summarizes the notation used by system dynamics modeling. The primary elements are variables of interest, stocks (which represent collection points of resources), and flows (which represent the transition of resources between stocks). Signed arrows represent causal relationships, where the sign indicates how the variable at the arrow’s source influences the variable at the arrow’s target. A positive (S) influence indicates that the values of the variables move in the *same* direction, whereas a negative (O) influence indicates that they move in *opposite* directions. A connected group of variables, stocks, and flows can create a path that is referred to as a feedback loop. There are two types of feedback loops: *balancing* and *reinforcing*. The type of feedback loop is determined by counting the number of negative influences along the path of the loop. An odd number of negative influences indicates a balancing loop; an even (or zero) number of negative influences indicates a reinforcing loop.

Significant feedback loops identified within the model described here are indicated by a loop symbol and a loop name in italics. Balancing loops—indicated with the label *B* followed by an identifying number in the loop symbol—describe aspects of the system that oppose change, seeking to drive variables to some equilibrium goal state. Balancing loops often represent actions that an organization takes to manage, or mitigate a problem. Reinforcing loops—indicated with a label *R* followed by a number in the loop symbol—describe system aspects that tend to drive variable values consistently either upward or downward. Reinforcing loops often represent the escalation of problems, but may include problem mitigation behaviors.



Variable – anything of interest in the problem being modeled.

Ghost Variable – variable acting as a placeholder for a variable occurring somewhere else

Positive Influence – values of variables move in the same direction (e.g., source increases, target increases)

Negative Influence – values of variables move in the opposite direction (e.g., source increases, the target decreases)

Delay –significant delay from when Var1 changes to when Var2 changes

Balancing Loop – a feedback loop that moves variable values to a goal state; loop color identifies circular influence path

Reinforcing Loop – a feedback loop that moves variable values consistently upward or downward; loop color identifies circular influence path

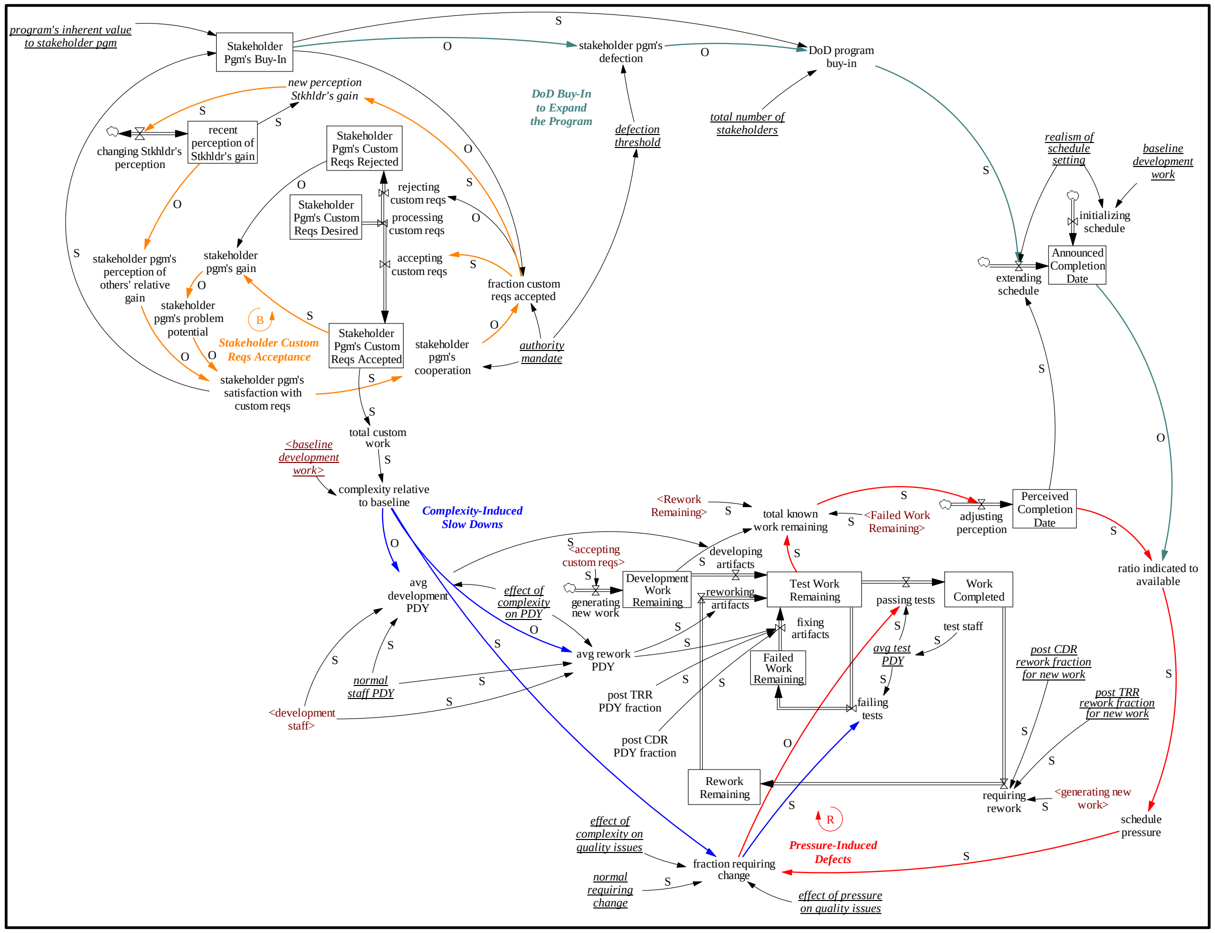
Stock – special variable representing a pool of materials, money, people, or other resources.

Flow – special variable representing a process that directly adds to or subtracts from a stock.

Cloud – source or sink (represents a stock outside the model boundary)

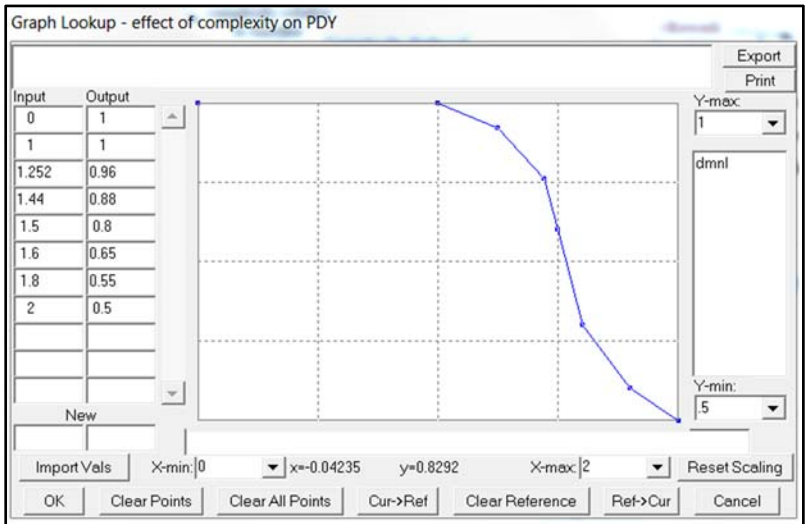
Figure 12. System Dynamics Notation

Appendix B: Overview of Joint Acquisition Program Model



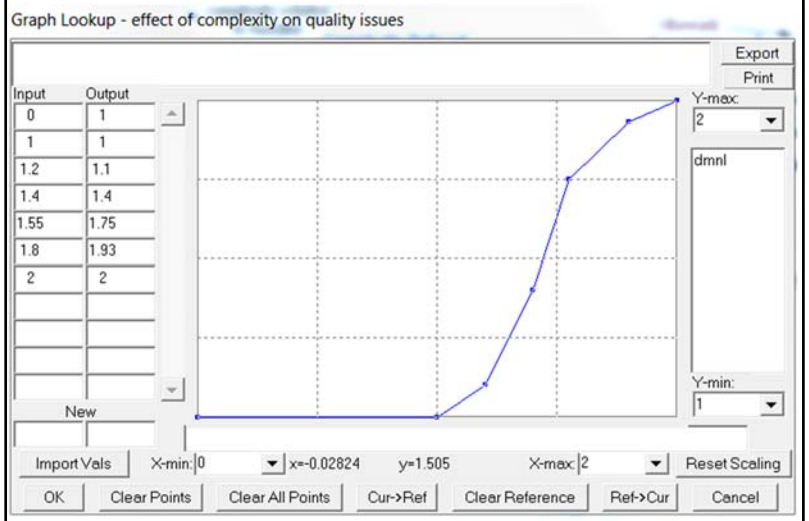
program's inherent value to stakeholder pgm
Stakeholder Pgm's Buy-In
stakeholder pgm's defection
DoD program buy-in
new perception Stkhldr's gain
changing Stkhldr's perception
recent perception of Stkhldr's gain
Stakeholder Pgm's Custom Reqs Rejected
DoD Buy-In to Expand the Program
defection threshold
total number of stakeholders
rejecting custom reqs
Stakeholder Pgm's Custom Reqs Desired
processing custom reqs
accepting custom reqs
stakeholder pgm's perception of others' relative gain
stakeholder pgm's gain
stakeholder pgm's problem potential
fraction custom reqs accepted
B
Stakeholder Custom Reqs Acceptance
Stakeholder Pgm's Custom Reqs Accepted
stakeholder pgm's cooperation
authority mandate
stakeholder pgm's satisfaction with custom reqs
total custom work
<baseline development work>
complexity relative to baseline
Complexity-Induced Slow Downs
realism of schedule setting
baseline development work
initializing schedule
extending schedule
Announced Completion Date
<Rework Remaining>
total known work remaining
<Failed Work Remaining>
adjusting perception
Perceived Completion Date
ratio indicated to available
<accepting custom reqs>
developing artifacts
avg development PDY
effect of complexity on PDY
generating new work
Development Work Remaining
reworking artifacts
Test Work Remaining
passing tests
Work Completed
fixing artifacts
test staff
avg rework PDY
Failed Work Remaining
avg test PDY
post CDR rework fraction for new work
normal staff PDY
post TRR PDY fraction
failing tests
post TRR rework fraction for new work
<development staff>
post CDR PDY fraction
Rework Remaining
requiring rework
<generating new work>
R
schedule pressure
effect of complexity on quality issues
Pressure-Induced Defects
fraction requiring change
normal requiring change
effect of pressure on quality issues
S
O

Appendix C: Development and Rework Effect Functions


Graph Lookup - effect of complexity on PDY

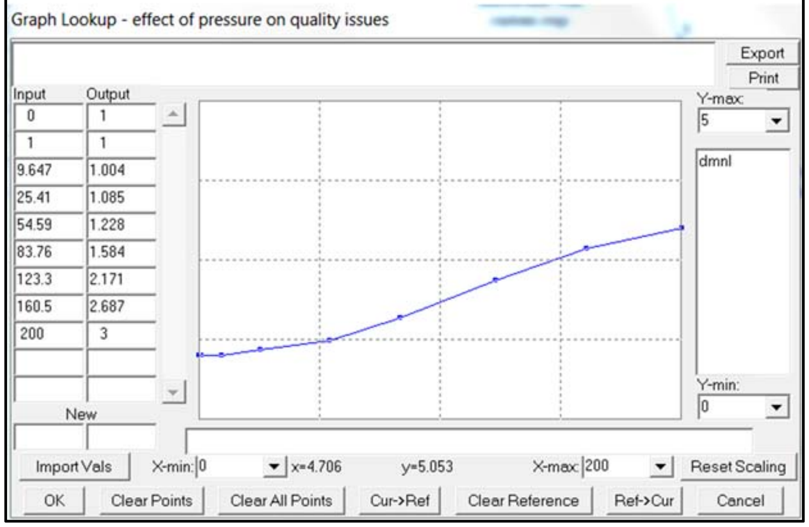
Input	Output
0	1
1	1
1.252	0.96
1.44	0.88
1.5	0.8
1.6	0.65
1.8	0.55
2	0.5

Y-max: 1
dmnl
Y-min: .5
X-min: 0 x=-0.04235 y=0.8292 X-max: 2


Graph Lookup - effect of complexity on quality issues

Input	Output
0	1
1	1
1.2	1.1
1.4	1.4
1.55	1.75
1.8	1.93
2	2

Y-max: 2
dmnl
Y-min: 1
X-min: 0 x=-0.02824 y=1.505 X-max: 2


Graph Lookup - effect of pressure on quality issues

Input	Output
0	1
1	1
9.647	1.004
25.41	1.085
54.59	1.228
83.76	1.584
123.3	2.171
160.5	2.687
200	3

Y-max: 5
dmnl
Y-min: 0
X-min: 0 x=4.706 y=5.053 X-max: 200

Acknowledgements

Many people have worked to sponsor and improve this paper and the work it describes, and it would not have been possible without their support, help, and expertise. First and foremost, we would like to thank our sponsor, the office of the United States Assistant Secretary of Defense (Research and Engineering) for the funding and opportunity to perform this work. Next, we would like to thank our group modeling workshop participants - the acquisition professionals who generously gave their time and expertise to help us to better understand the problem and improve the model formulation. We would also like to thank Linda Northrop, Chief Scientist of the SEI's Software Solutions Division, for her support, encouragement, and insights. We are grateful to James McCurley, Senior Member of Technical Staff at the SEI, who provided useful guidance and direction throughout our data analysis and modeling efforts. Finally, the authors owe a debt of gratitude to our editor, Gerald Miller.