Modeling the Evolution of a Science Project in Software-Reliant System Acquisition Programs

Andrew P. Moore, apm@cert.org, 412-268-5465
William E. Novak, wen@sei.cmu.edu, 412-268-5519

Software Engineering Institute
4555 Fifth Avenue
Pittsburgh, PA 15213

Abstract

Analysis work by the Software Engineering Institute (SEI) on data collected from more than 100 Independent Technical Assessments (ITAs) of software-reliant acquisition programs has produced insights into some of the most common ways that programs encounter difficulties. This paper describes work done at the SEI that is based on these insights, and intends to mitigate the effects of both misaligned acquisition program organizational incentives, and adverse software-reliant acquisition structural dynamics, by improving acquisition staff decision-making. The research presented here uses a preliminary system dynamics model to analyze a specific adverse acquisition dynamic concerning the poorly controlled evolution of small prototype efforts into full-scale systems that is called "The Evolution of a Science Project." The system dynamics model of the behavior is described, along with the results of simulations run using the model. The paper concludes with a set of lessons learned, as well as potential future research directions.

1 Introduction

Software-reliant systems are those that rely on software to provide their core mission functions—and the failure of programs developing these systems is an all too common occurrence for modern government acquisition. The Software Engineering Institute (SEI) is periodically asked to investigate the reasons why some software-reliant acquisition programs are seriously challenged, and even fail, despite the best efforts by government and industry to make them successful. The SEI works closely with, and in many cases conducts assessments of, individual software-reliant acquisition programs that are facing such challenges. These engagements have provided visibility into the processes and forces at work within these programs. Analysis work by the SEI on data collected from over 100 Independent Technical Assessments (ITAs) of software-reliant acquisition programs has produced insights into some of the most common ways that programs encounter difficulties. At a high level it has become evident that acquisition programs continue to regularly experience recurring cost, schedule, and quality failures, and program progress and outcomes often appear to be unpredictable and unmanageable (U.S. GAO, 2005).

Furthermore, many acquisition leaders and staffers neither recognize these issues (despite their regular recurrence), nor understand how to manage them (Kadish, 2006). Despite best efforts by the Department of Defense to train and equip the acquisition workforce with the skills and guidance it needs to manage large, complex programs, significant challenges remain. Government and industry have an urgent need to help the defense acquisition workforce and their defense industry counterparts better recognize and understand the reasons behind the failures of many acquisition programs, and to provide guidance on how to mitigate or avoid these problems in the pursuit of better outcomes.

One growing theme in acquisition is the prevalence of programs that are sometimes called "science projects," (which may take the form of rapid prototyping efforts, advanced research projects, or Advanced Concept Technology Demonstrations) and the difficulties associated with evolving such efforts into larger and more formal acquisition programs. The name “science project” refers to a program that starts out small and informally, often as an experimental development or prototype system to show proof of concept, and then grows in scope as demand increases. This paper describes a commonly recurring pattern of behavior seen by the SEI in software-reliant acquisition programs called “The Evolution of a Science Project,” and its analysis using system dynamics modeling and simulation. This pattern was identified over the course of multiple ITAs that gathered qualitative data on the behavior, and was endorsed as being timely and relevant in the current acquisition environment by both acquisition practitioners and executives.

This paper provides an overview of the results developed in the initial year of work devoted to this effort. The purpose of this effort is to a) model and understand a timely and relevant dynamic that recurs frequently in software acquisition programs, and b) create a foundation of expertise and system dynamics modeling infrastructure that can be built upon in subsequent work. The concept of rework, which is a central idea behind the dynamics of “The Evolution of a Science Project,” is a major part of most project management models. Rework refers to the production of work in which some portion of the work has been done incorrectly, and is then classified as either discovered or undiscovered (i.e., latent) rework which will need to be performed again, often requiring unplanned effort. Rework is also sometimes referred to as “firefighting,” in the sense that as defects are inserted into the system, developers may be diverted to fix them, thus slowing progress and increasing schedule pressure, which can shortcut quality assurance processes, and so inject more defects.

Project management dynamics have been studied using system dynamics modeling for over thirty years. Seminal works in the area include:

- Ken Cooper, “Naval Ship Production: A Claim Settled and a Framework Built” (Cooper, 1980)
- Tarek Abdel-Hamid and Stuart Madnick, *Software Project Dynamics* (Abdel-Hamid & Madnick, 1991)
- David Ford and John Sterman, “Dynamic Modeling of Product Development Processes” (Ford & Sterman, 1998)
- Raymond Madachy, *Software Process Dynamics* (Madachy R. J., 2008)

Rework continues to be the subject of research in system dynamics (Li, Taylor, & Ford, 2011) (Li Y. , 2011) (Taylor, Ford, & Johnson, 2005) (Lyneis & Ford, 2007). In addition, rework is inherent in software development, often resulting in substantial cost and schedule growth, in part because of the feedback that it presents. Without tooling such as system dynamics modeling that can be used in its study, rework presents a complex and nonlinear problem that is generally intractable to traditional analysis methods. Software-specific research involving rework involves work in the area of software development projects (Birkholzer, et al., 2010) (Raffo & Kellner, 2000) (Scacchi & Mi 1993) (Madachy, Boehm, & Lane, 2006, May), and, to a lesser extent, software acquisition (Wirthlin, Houston, & Madachy, 2011) (Haberlein, 2004) (Abelson, Adams, & Eslinger, 2004, January) (Scacchi & Choi, 2001) (Buettner, 2008) (Choi & Bae, 2006).

The key questions that this work is attempting to answer are as follows:

- Can the management and software development aspects of software-reliant acquisition programs be cost-effectively modeled using system dynamics such that the simulation results reasonably and consistently conform to the outcomes seen in actual programs?
- Can the results of such modeling be used to produce practical insights into different ways to improve the conduct and management of software-reliant acquisition programs?

This paper characterizes our hypothesis regarding the problem regarding system acquisitions that involve the development of an early science project to explore key issues. The causal structure underlying our hypothesis is described as a causal loop diagram, which in section 3 is refined into a full system dynamics simulation model. The next section outlines some of the preliminary results of the simulation with key observations that support our hypothesis. The last section summarizes the paper, describes some key lessons learned through our modeling effort, and describes remaining work to be done. The appendices provide an overview of the system dynamics method and notation, the models developed, and an interface used to interact with the model simulation. This paper is a summary of the full report (Novak, Moore, & Alberts, 2012).

2 Problem Characterization

We hypothesize that the root cause for the problematic behavior experienced during the transition of many science projects is due to the fact that

- software development of the prototype is driven by schedule and the development of new features, with little focus on quality and robustness.
- the code from the science project prototype is used as the foundation for the production system because of the perception that starting over would unacceptably delay deploying new capabilities to the field (the architecture and design of the prototype might also be used, but in many cases these do not exist, and when they do, they are frequently poorly documented).
- the transition from prototype development to production system development is delayed later than necessary, both to try to field a system more quickly, and perhaps consciously, to avoid the effort and overhead of formal program rigor.

Early in the life cycle of a science project, development results in a working prototype that has attractive external capabilities, but also high defect rates, poor code quality (i.e., complex and poorly structured code), and poor system documentation. The use of the prototype as the basis for a production system and the late transition of the prototype to production development inject a significant level of latent defects into the system. When discovered incrementally later in development, the latent defects and poor code quality lead to a “ripple effect” of rework across the phases of the development (Cooper, 1980) (Taylor, Ford, & Johnson, 2005, July). This creates schedule pressures in the production development that can magnify the ripple effect further. The ripple effect is depicted in Figure 1 below, notionally graphing discovered quality issues during production development over time. Discovered quality issues rapidly increase to a peak, in many cases leading over time to additional consequential issues—and then gradually decline as development resources are reallocated to rework them, with this rework itself introducing a smaller level of quality issues that will have to be reworked.

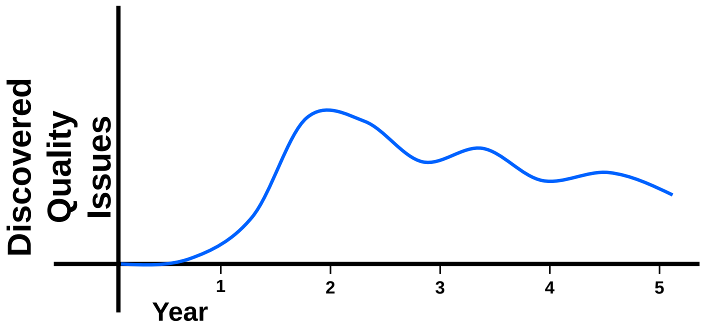


Figure 1: Notional Problem: Ripple Effect Due to Incremental Discovery of Science Project Quality Issues

We believe that the ripple effect described above can cause the “90% syndrome,” as depicted in Figure 2, where latent defects that have accumulated during the science project phase slowly come to light during the production development in the later stages of the project. During the latter stages of development a much larger percentage of the effort is devoted to rework rather than development. That situation, combined with the fact that undiscovered work comes to light only incrementally over the lifetime of the project, results in the slowed progress inherent in the 90% syndrome.

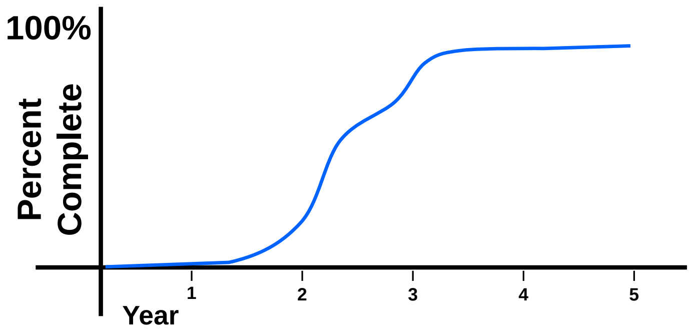


Figure 2: 90% Syndrome Due to Rippling Rework in the Production Development

Figure 3 (depicted on the next page) presents a causal loop diagram[1] that refines the causal hypothesis described above. The left side depicts the primary aspects of the development of the science project prototype. The right side shows the transfer of that prototype to production development. The development of the science project is motivated by urgent demand for the new capabilities in the field. The purpose is to build a small prototype system to produce a compelling “proof of concept” demonstration to help garner support for full-scale production development.[2] The project is often (although not always) initiated and managed by the user or operational community. The software development is often done in-house by existing staff, and if a contractor is used the contract vehicle may be Time and Materials (T&M) or Indefinite Delivery/Indefinite Quantity (IDIQ) without explicit schedules or deliverables, in order to keep the development effort as informal as possible. The effort is generally both schedule and feature driven to get key functionality to users as quickly as possible. The (blue) feedback loop on the left side of Figure 3 shows that schedule pressure can result in a reduction in the rigor of quality assurance (QA) processes, with a commensurate reduction in the staffing of rework to fix quality issues in the prototype. This results in more effort being put into developing features and a reduction of schedule pressure compared to what it would have been otherwise.

[1] A causal loop diagram shows qualitatively how related variables affect each other. The nodes indicate variables, and the connecting arrows show the nature of the relationships between them. When an increase in one variable causes an increase in the variable it is connected to, the connecting arrow is marked with a “+”. When a change in one variable causes the *opposite* change in the variable it is connected to, the connecting arrow is marked with a “-“. A loop made up of nodes and connecting arrows that continually moves in the same direction is called a “reinforcing” loop. A loop where the values of the nodes oscillate back and forth is called a “balancing” loop. Two parallel lines placed across a connecting arrow indicate that there is a delay in the propagation of the value across the arrow. A more detailed discussion of the notation for causal loop diagrams may be found in Appendix B of this paper.

[2] It should be noted again that this scenario is *not* what would be done when following an evolutionary prototyping approach in support of agile development method, where formal steps would be taken to ensure the quality and robustness of the evolving prototype.

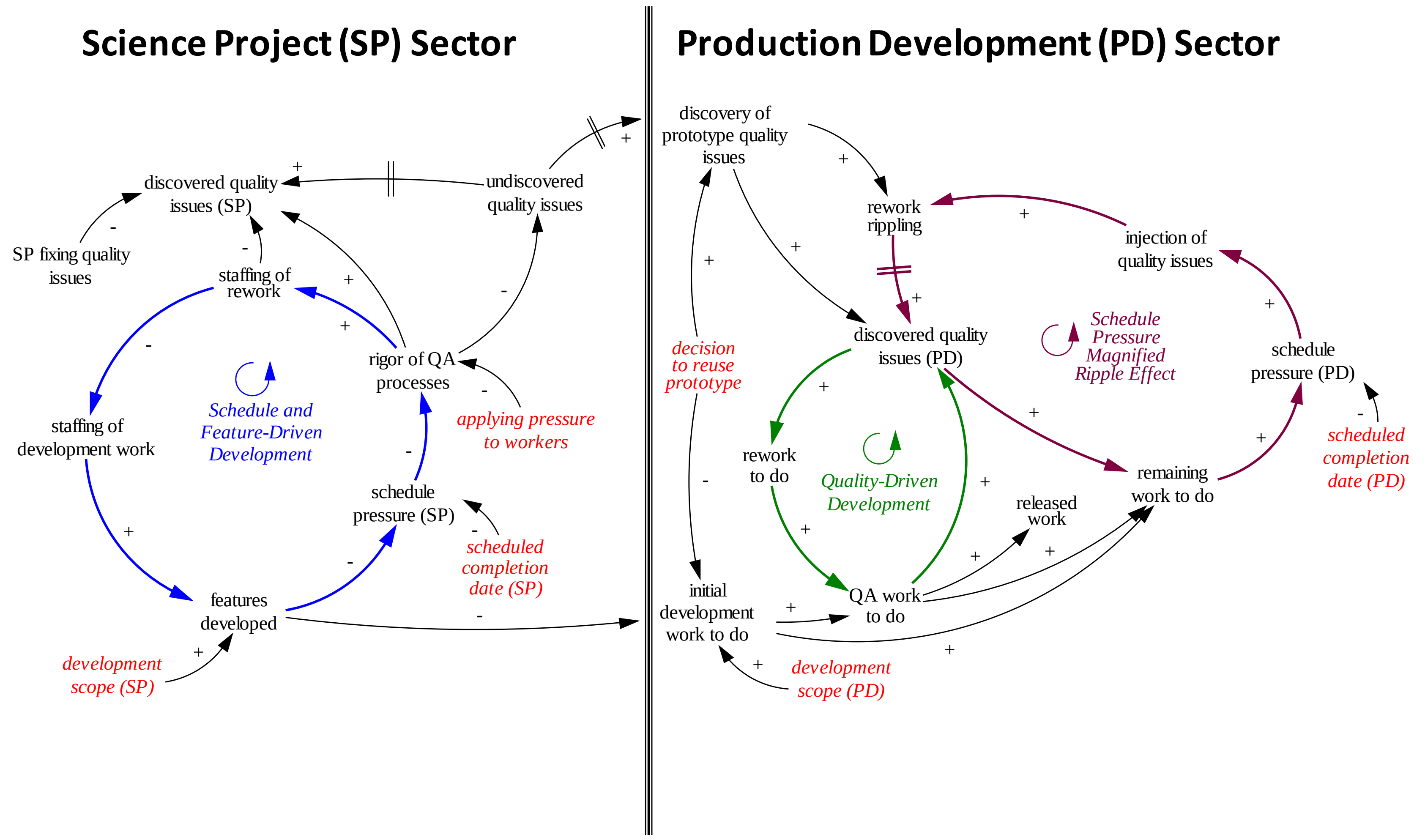

Science Project (SP) Sector
Production Development (PD) Sector
discovered quality issues (SP)
undiscovered quality issues
SP fixing quality issues
staffing of rework
rigor of QA processes
staffing of development work
Schedule and Feature-Driven Development
applying pressure to workers
schedule pressure (SP)
scheduled completion date (SP)
features developed
development scope (SP)
discovery of prototype quality issues
rework rippling
injection of quality issues
discovered quality issues (PD)
Schedule Pressure Magnified Ripple Effect
schedule pressure (PD)
decision to reuse prototype
rework to do
Quality-Driven Development
released work
remaining work to do
scheduled completion date (PD)
initial development work to do
QA work to do
development scope (PD)

Figure 3: Causal Loop Diagram of "The Evolution of a Science Project"

While reducing the rigor of the QA processes enables more rapid development of features, it also has clear repercussions on the quality of the prototype as depicted in the upper portion of the Science Project sector. Reduced QA rigor leads to the generation of more software quality issues, which are initially unknown (or undiscovered) by the project team. It also makes it difficult to discover and fix these quality issues (fewer QA inspections, etc.) when staffing of rework is diminished. The result is that quality issues are rampant in the prototype when the decision is made to transition the system to a formal acquisition program, especially when the project resists making that transition as soon as possible after proving viability. This dynamic is supported by and consistent with a ripple effect exhibited in previously developed project management models (Taylor, Ford, & Johnson, 2005).

The decision to reuse the prototype in production development may be irresistible. As was pointed out previously, having operational people manage the project instead of software development or acquisition professionals may prevent them from seeing the longer-term consequences of using a prototype as the basis for production development. Upon the decision to reuse the prototype, as shown on the right side of the Production Development sector, the production development inherits the work that has been completed on the prototype. Unfortunately, it also inherits the prototype's quality issues. Most significant is the large stock of undiscovered quality issues. While the production development adopts a quality-driven development approach appropriate for a formal acquisition program, as shown in the (green) feedback loop, the discovery of prototype quality issues creates a ripple effect of rework across the phases of the production development.

Of course, the production development effort is not immune to its own schedule pressure. While some level of defects was expected, the incremental discovery of latent prototype quality issues stemming from significant architectural and design flaws was not fully considered in the schedule for the production development. The large amount of rework caused by the ripple effect of the incrementally discovered quality issues throws the production development off schedule, thus increasing pressure on development staff and as a consequence, increasing the injection of quality issues during development. This effect magnifies the ripple effect further as seen in the (purple) feedback loop on the right side of Figure 3.[3]

It might appear that if the program management simply did a better job of planning for the additional rework caused by the prototype, the problems could be resolved. However, it is the existence of fundamental issues with the prototype software's architecture and design that led to much of the undiscovered (and unsuspected) rework that will have to be done—and the effort that will be involved to address those issues will be both substantial and especially difficult to estimate. The conclusion is that the causal mechanisms described above may be the determining factor behind the poor performance of many science projects even after they have transitioned to a formal acquisition program.

3 The Simulation Model

A quantitative system dynamics model refines and describes the relationships presented in the causal loop diagram using mathematical equations. This is done by adding two new concepts to the modeling notation: stocks and flows.

1. Stocks represent accumulations of physical or non-physical quantities, and flows represent the movement of these quantities between stocks. Stocks are depicted as named boxes within the model.
2. Flows are depicted as double-lined arrows between the stocks with a named valve symbol indicating the name of the flow. Flows that come from (or go to) a cloud symbol indicate that the stock from

[3] Note that in the diagram, QA Work to Do has the same influence on both released work, and remaining work to do. This is simply a reflection of the fact that as the amount of QA work increases, it results in both more released work, and more remaining work to do (i.e., rework).

which the flow originates (or to which the flow goes) is outside the scope of the model, and presumed to be infinite.

Figure 4 provides an overview of the system dynamics model. The three feedback loops used to describe the causal hypothesis in the last section also appear in this overview. The loops correspond with loops B2, R3, and R4 in the figure and are labeled with the same name and appear in the same color. The left side of the figure portrays the Science Project sector, and the right side shows the Production Development sector. The overview in Figure 4 provides an index to the structure of the full model presented in Appendix C and Appendix D.

The overview presents both the primary and secondary feedback loops that are most important for describing the evolution of a science project. At this level there are nine feedback loops—five balancing loops and four reinforcing loops.[4] The model embodies a simplified story of the evolution of a science project, which integrates the exposition of the feedback loops described in more detail in the next section. A more detailed version of this story, drawn from assessments of acquisition programs, can be found in the full version of the report. (Novak, Moore, & Alberts, 2012)

Developers build features while reworkers fix quality issues. Developers may get pulled off development to do quality driven rework (feedback loop ***B1****).*[5] *However, science projects are largely schedule and feature-driven to satisfy users, so managers resist staffing rework at sufficient levels (****B2****). While new features may ultimately satisfy users (****B3****), in the short term they create heavy user demand, increasing schedule pressure (****R1****). Increased schedule pressure may increase short-term productivity (****B4****), but at the expense of more quality problems. Even worse, workers eventually start to burn out due to the increased work load (****R2****). The steady erosion of the QA processes and increasing problems in the field creates pressure to move to a formal acquisition program.*

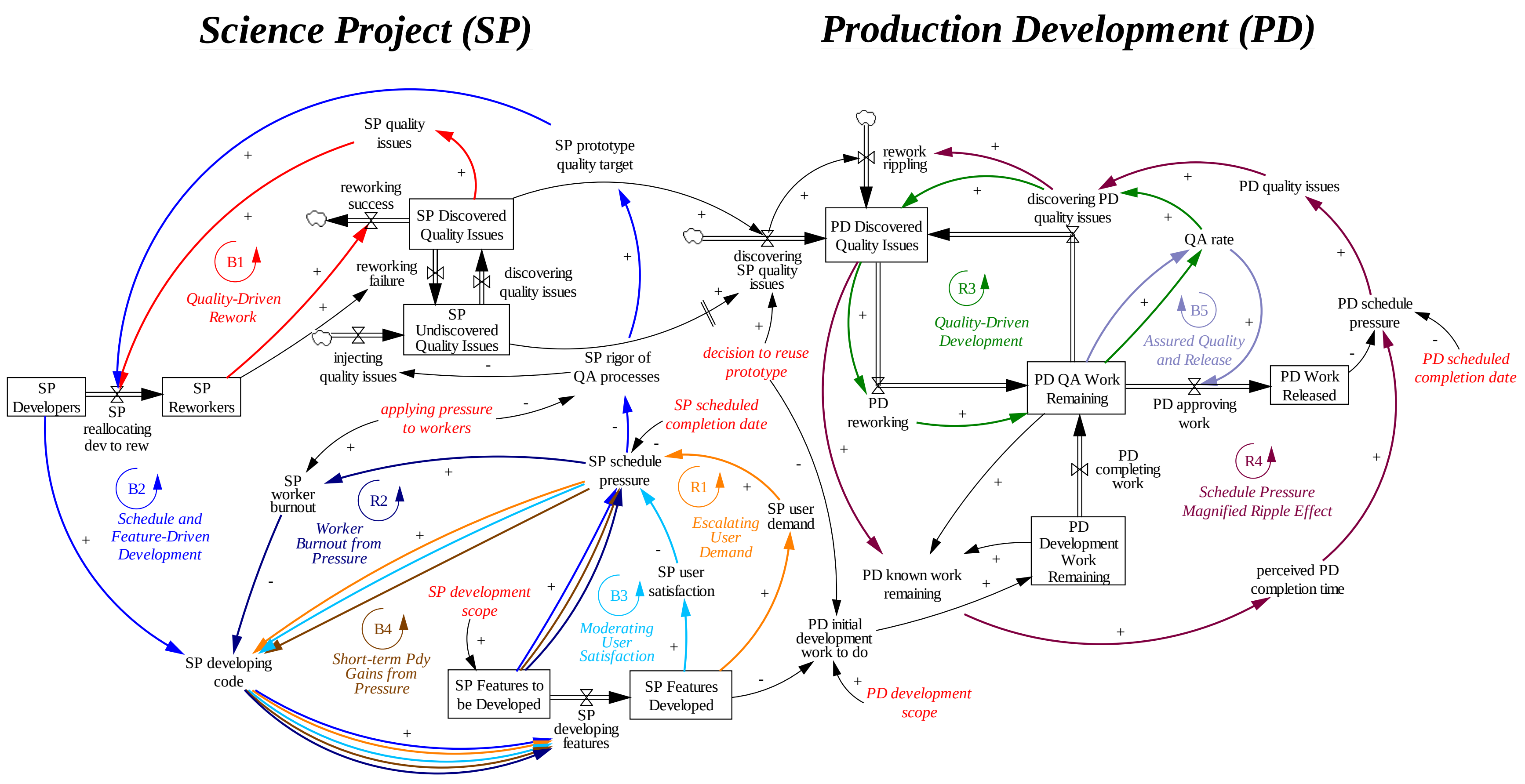
As the science project transitions to production development, that development inherits undiscovered quality issues in the science project prototype.[6] *This creates a "ripple effect" of rework that is discovered incrementally through the production development (****R3****). All the while, the production development strives to approve and release work to increase the capability of the fielded system (****B5****). Unfortunately, schedule pressure during the production development mounts due to the incremental discovery of pre-existing quality issues (and thus additional rework) from the science project. The ongoing pressure creates quality problems in the production development as well, and magnifies the ripple effect of undiscovered quality issues (****R4****).*

---

[4] The terms "balancing loops" and "reinforcing loops" are described in Appendix B as part of an introduction to system dynamics.

[5] Note in the diagram that more SP Reworkers results in an increase in both reworking success *and* reworking failure. This is a reflection of the fact that as the number of SP Reworkers increases, discovered quality issues will be dealt with more quickly through rework done in either a correct (successful) or incorrect (failing) manner. While seemingly counter-intuitive, this is simply a consequence of the increased amount of work being done. We assume that there is no change in the percentage of rework success due to an increase in SP Reworkers.

[6] While it looks like SP Discovered Quality Issues have to be rediscovered in the production development, the quality issues that are discovered during the science project development are immediately available as discovered quality issues in the production development, while the undiscovered quality issues take some time to discover during production development. This is signified by the delay mark on the arrow from SP Undiscovered Quality Issues to PD Discovered Quality Issues. This behavior is apparent in the mathematical formulas representing the simulation model, but is somewhat ambiguous at this qualitative level.



Science Project (SP)
Production Development (PD)
SP quality issues
SP prototype quality target
reworking success
SP Discovered Quality Issues
reworking failure
discovering quality issues
SP Undiscovered Quality Issues
injecting quality issues
SP rigor of QA processes
B1
Quality-Driven Rework
SP Developers
SP reallocating dev to rew
SP Reworkers
applying pressure to workers
SP scheduled completion date
SP schedule pressure
SP worker burnout
B2
Schedule and Feature-Driven Development
R2
Worker Burnout from Pressure
R1
Escalating User Demand
SP user demand
SP user satisfaction
B3
Moderating User Satisfaction
SP development scope
B4
Short-term Pdy Gains from Pressure
SP developing code
SP Features to be Developed
SP developing features
SP Features Developed
discovering SP quality issues
decision to reuse prototype
rework rippling
PD Discovered Quality Issues
discovering PD quality issues
PD quality issues
QA rate
R3
Quality-Driven Development
B5
Assured Quality and Release
PD schedule pressure
PD scheduled completion date
PD reworking
PD QA Work Remaining
PD approving work
PD Work Released
PD completing work
R4
Schedule Pressure Magnified Ripple Effect
PD Development Work Remaining
PD known work remaining
PD initial development work to do
PD development scope
perceived PD completion time

Figure 4: Overview of the System Dynamics Model

3.1 Science Project Sector

The key stocks and flows in the Science Project (SP) sector are depicted in Figure 5 and are understood as follows:[7]

- *Worker reallocation (along the left middle of Figure 5)*: Science project personnel are either developers, i.e., writers of code to implement features; or reworkers, i.e., fixers of quality issues related to code defects, code quality, or software architecture. We assume a fixed number of employees working on the science project prototype, but with full flexibility to move staff between development work and rework based on the priorities of the project, with a small time lag to actually make the transition. Note that while the flow is only shown by the arrow to occur in one direction, a negative flow results in reworkers becoming developers.
- *Feature development (along the bottom right of the figure):* The model starts out with a basic number of features to be developed. The stock and flow shows that features are developed at a rate represented by the "SP developing features" variable. In the model this rate is based on the number of SP Developers available and their (average) productivity,
- *Quality issue processing (along the top right of the figure):* Quality issues injected into the software during development are initially unknown by science project personnel. Over time these quality issues may be discovered and, depending on the number of SP reworkers available, the associated software will be reworked. The rework may be successful, resulting in the correction of the problem, or may fail, leading to additional undiscovered quality issues.

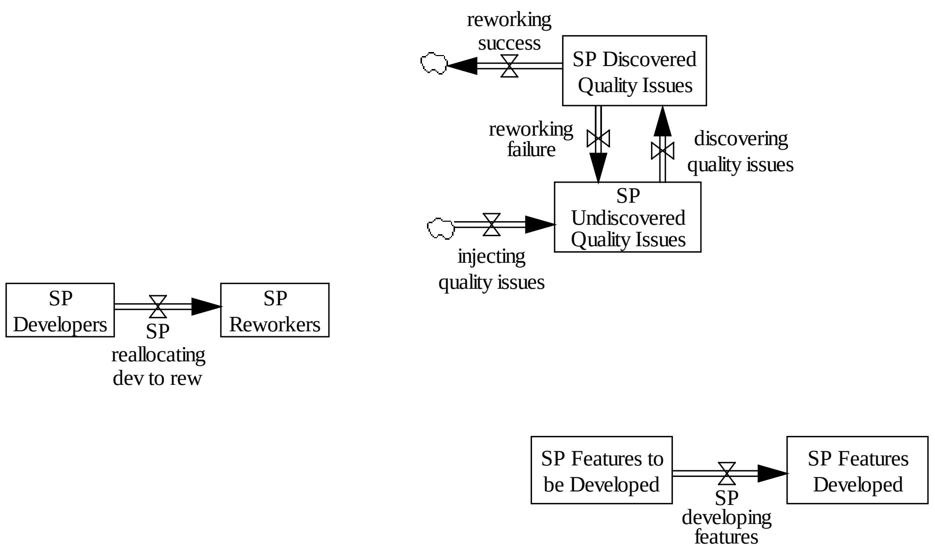

reworking success
SP Discovered Quality Issues
reworking failure
discovering quality issues
SP Undiscovered Quality Issues
injecting quality issues
SP Developers
SP reallocating dev to rew
SP Reworkers
SP Features to be Developed
SP developing features
SP Features Developed

Figure 5: Stocks and Flows in the Science Project Sector

[7] While there are additional stocks and flows in the full model shown in the appendices, we believe that the ones describe here are key to understanding the basic dynamics of the model. By understanding this simpler model, the interested reader will be well on their way to understanding the full model.

The stocks and flows of Figure 5 are interconnected via the feedback loops shown in the Science Project Sector of Figure 4. The key feedback loops are:

- *B1: Quality-Driven Rework (red)*: This balancing loop represents the relationship in the Science Project between the number of people doing rework, and the number of discovered quality issues in the science project. It shows the feedback in attempting to reduce quality issues through reallocating people to do rework.
- *B2: Schedule and Feature-Driven Development (medium blue)*: This balancing loop represents the relationship in the Science Project of the number of developers, the code and features they develop, schedule pressure, QA rigor, and the prototype quality target. It represents balancing feedback in attempting to meet schedule by lowering QA standards, reallocating people to do development rather than rework (i.e., sacrificing quality to make better progress), produce more features, and reduce schedule pressure.
- *R1: Escalating User Demand (orange)*: This reinforcing loop represents the relationship in the Science Project between the number of features that have been developed, the user demand for more features, schedule pressure from users, and the amount of developed code. It shows the feedback effect of how more features having been developed actually increases the user demand for even more features, driving up schedule pressure to develop more code.
- *B3: Moderating User Satisfaction (aqua)*: This balancing loop represents the relationship in the Science Project that tends to reduce (somewhat) the schedule pressure, as users get access to features developed by the Science Project and to some extent are satisfied with those features. This loop, therefore, partially counteracts the R1 loop described above. Combined with the R1 loop, the cumulative result may still be that schedule pressure increases based on user reactions, but the B3 loop represents those influences that dampen the schedule pressure below what it would have been otherwise.
- *B4: Short-term Productivity Gains from Pressure (brown)*: This balancing loop represents the relationship in the Science Project between the amount of developed code, the number of new features developed, user satisfaction with those features, and schedule pressure from users for more features. It shows the feedback in attempting to boost productivity by pressuring developers to produce more code, thus relieving the schedule pressure at least in the short term.
- *R2: Worker Burnout from Pressure (navy blue)*: This reinforcing loop represents the relationship in the Science Project between the amount of developed code, the number of features developed, user satisfaction with those features, schedule pressure from users for more features, and worker burnout from the schedule pressure. It shows the feedback effect of ongoing schedule pressure producing burnout among developers, thus adversely impacting their productivity in developing code and increasing schedule pressure.

3.2 Production Development Sector

The key stocks and flows in the Production Development (PD) sector are depicted in Figure 6. The stocks are

- *PD Development Work Remaining*: This stock contains the development work to be done. Once this work is completed (at whatever level of quality the current processes afford) the work is sent on for quality assurance evaluation.
- *PD QA Work Remaining*: As development work is completed, quality assurance processes take over. This stock contains the backlog of QA work to be done.
- *PD Work Released*: Work that is approved is released into the PD Work Released stock. How the work is used once it is released is not modeled, nor is allowance made for the possible discovery of

quality issues in released work products. For this model we assume that the quality assurance processes are sufficient to ensure acceptable use of the work products in the field.

- *PD Discovered Quality Issues*: Work that is found to have significant quality issues goes to the PD Discovered Quality Issues stock. Quality issues may also be discovered due to problems in the SP prototype, which has a separate flow into the stock of discovered quality issues. The rework rippling flow is yet another inflow to discovered quality issues that represents the potential for ripple effects of additional rework due to quality issues found in previously developed artifacts.

The stocks and flows of Figure 6 are interconnected via the feedback loops shown in the Science Project Sector of Figure 4. This leverages insights from the project management model of (Taylor, Ford, & Johnson, 2005). The key feedback loops are as follows:

- *B5: Assured Quality and Release (gray)*: This balancing loop represents the relationship in Production Development between the amount of QA work remaining, the rate at which QA work is being accomplished, and the rate at which work is being approved. It shows the feedback of pending QA work that influences the rate of approving completed work.
- *R3: Quality-Driven Development (green)*: This reinforcing loop represents the relationship in Production Development between the amount of QA work remaining, the rate at which QA work is being accomplished, the discovery of quality issues, the rate at which rework is being done on those issues. It shows the feedback effect in which the discovery of quality issues as a result of the QA process causes rework, thus increasing the QA backlog in a reinforcing way. This is one dimension of the ripple effect of undiscovered quality issues.

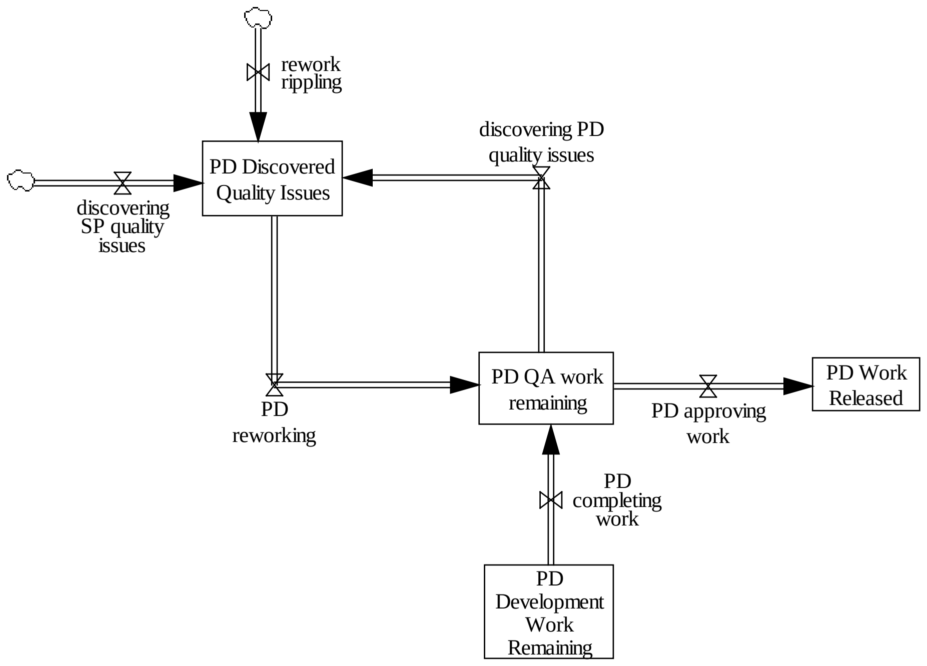

rework rippling
discovering PD quality issues
PD Discovered Quality Issues
discovering SP quality issues
PD QA work remaining
PD Work Released
PD reworking
PD approving work
PD completing work
PD Development Work Remaining

Figure 6: Stocks and Flows in the Production Development Sector

- *R4: Schedule Pressure Magnified Ripple Effect (purple)*: This reinforcing loop represents the relationship in Production Development between the amount of known work that is remaining to be done, the perceived time needed to finish that work, the schedule pressure, quality issues, and the discovery of quality issues. It shows the feedback effect in which discovered quality issues increases schedule pressure, and magnifies the ripple effect described in the R4 loop. We assume that the additional rework generated as a result of the ripple effect is 0.8 times the rework due to discovered

PD and SP quality issues. This multiplier is the ripple effect strength referred to in (Taylor, Ford, & Johnson, 2005).

Note that the rework (or "Firefighting") dynamic is evident in the Science Project sector in the *B2: Schedule and Feature-Driven Development* feedback loop, and in the Production Development sector in the *R4: Quality-Driven Development* and *R5: Schedule Pressure Magnified Ripple Effect* feedback loops.

4 Preliminary Simulation Results

The full model (which is described in the appendices, and was discussed in overview in the previous section) is implemented to provide a simulation of the entire science project evolution. Appendix D shows the interface that was created to interact with the model. This section presents a few preliminary simulation behaviors that reflect the team's experience with science projects in the field at a qualitative level. The default initial value of the primary input variables are as follows:

- Applying pressure to workers: This variable indicates how much managers apply pressure to workers according to how behind schedule the project is. The default initial value is set to High.
- Development scope (SP): This variable regulates how many features are being implemented as part of the science project effort. The default initial value is set to High, which translates to 40 features. The complete development effort for the production development is 200 features.
- SP scheduled completion date: This variable indicates the date currently scheduled for completion of the science project effort. The default initial value is set to 90 weeks from the start of the simulation.
- decision to reuse prototype: This variable indicates whether the production development effort will use the artifacts produced as a result of the science project development effort. The default initial value is set to false (the value 0 represents false).

A word of caution is necessary. The model has not yet been validated or calibrated to align with the experience of acquisition program decision-makers outside the modeling team. As such this simulation model is only a preliminary explanation of the dynamics of "The Evolution of a Science Project." The causal hypothesis described previously remains a hypothesis. The simulation model described here provides one avenue for testing mitigations to the problematic behavior, but work remains in terms of refining and validating the quantified relationships among model variables before testing mitigations would have significance in the real world. The model must exhibit the *right behavior*, and for the *right reasons***,** before firm conclusions can be drawn from the simulation. Nevertheless, the simulation model presented in this paper provides tentative support for the causal hypothesis presented earlier. We therefore present the findings below as preliminary.

Preliminary Finding 1: The accumulation of undiscovered rework from the science project creates a tipping point during production development.

A tipping point is a threshold condition that, when crossed, shifts the dominance of the feedback loops controlling a process (Sterman, 2000). In our case, the shift is from the dominance of the balancing loop *B5: Assured Quality and Release* to the reinforcing loop *R4: Schedule Pressure Magnified Ripple Effect*. When B5 dominates, progress on the project is made, however steadily, towards completion. When R4 dominates, progress nearly halts as the ripple effect causes the generation of more and more rework.

Preliminary Finding 2: Applying high pressure, or even moderate pressure for extended periods, can push the program past the tipping point.

The tipping point dynamic is clearly seen in the simulation graph presented in Figure 7.[8] The degree to which schedule pressure is applied to workers is measured on a six-point scale: None, Low, Medium-Low, Medium, Medium-High, or High. During times of high schedule pressure, applying the full pressure to the workers might mean that the workers' jobs would be in jeopardy if they did not put forth their full effort, whereas medium pressure might mean that their performance evaluations would be significantly impacted. The graph shows that applying pressure at Medium level and above pushes the discovery (and generation) of quality issues over the tipping point towards more and more rework (this is where loop R4 dominates). Applying pressure at a Medium-Low level and below results in completion of the project (this is where loop B5 dominates).[9]

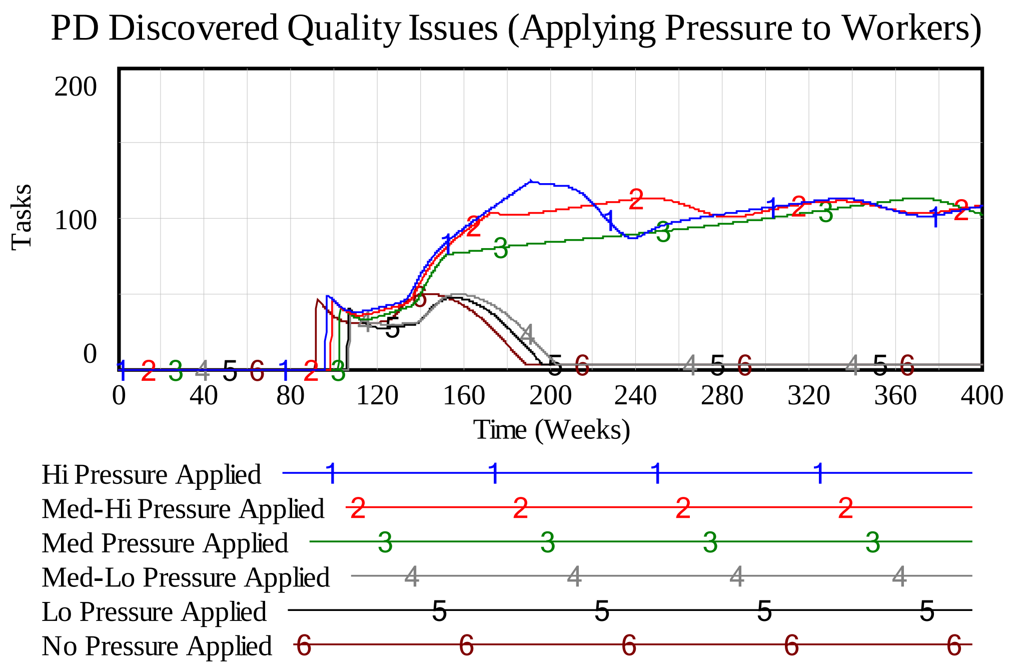


Figure 7: Simulation Results for Production Development Discovered Quality Issues

The above behavior is even worse than what we expected as described in the causal hypothesis and as shown in Figure 1. Not only does the ripple effect delay the completion of the project, but in the worst case it extends it out indefinitely toward more and more rework. Of course, if the project were to restart itself on solid ground it could again begin to make solid progress toward definite completion. However, the simulation suggests that if the project were left to run its course, no conclusion would be reached, indicating that the project was past the tipping point. Notice also the increasing amplitude of the (damped)

[8] This and subsequent graphs were generated using the Vensim modeling and simulation tool (VenSim is a registered trademark of Ventana Systems Inc.). These are all behavior-over-time graphs and, as such, the X-axis for these graphs is specified in weeks (400 weeks is the duration of the simulation). Each simulation run is specified as individual graphs distinguished both in color and with a number label (1 through 6 in Figure 7), as specified in the legend below the graph. Each simulation run varies a key parameter. The graph in Figure 7 varies the extent of Applying Pressure to Workers from no pressure applied up to high pressure applied, and shows the value of PD Discovered Quality issues along the Y-axis for all times during the simulation. The label on the Y-axis indicates the units for the variable being graphed, which in this case is the number of tasks involved with fixing the quality issues that were discovered.

[9] The simulation runs for PD Discovered Quality Issues at the lower pressure levels (runs 4, 5, and 6) shown in the figure do not actually show the levels going to zero in the second half of the simulation. The project does terminate, but some minimal level of rework is allowed to remain even at project conclusion, i.e., not all loose ends are completely tied up.

oscillations apparent when higher levels of pressure are applied. These oscillations in the model simulation occur precisely because of the higher levels of undiscovered rework transferred to the production development, as seen below in Figure 8.

The reason that all runs have no discovered quality issues up until between week 80 and week 120 is that this is the point at which the science project ends and the quality issues from the prototype are transferred to the production development. The graphs in Appendix F of the full report show the behavior of the science project prototype development prior to the point of this transfer for the case in which all pressure is applied (Novak, Moore, & Alberts, 2012). Key to the buildup of the initial PD Discovered Quality Issues is the science project's response to the schedule pressure by reducing the rigor of QA processes, and the resulting poor quality work that the schedule and feature-driven development entails.

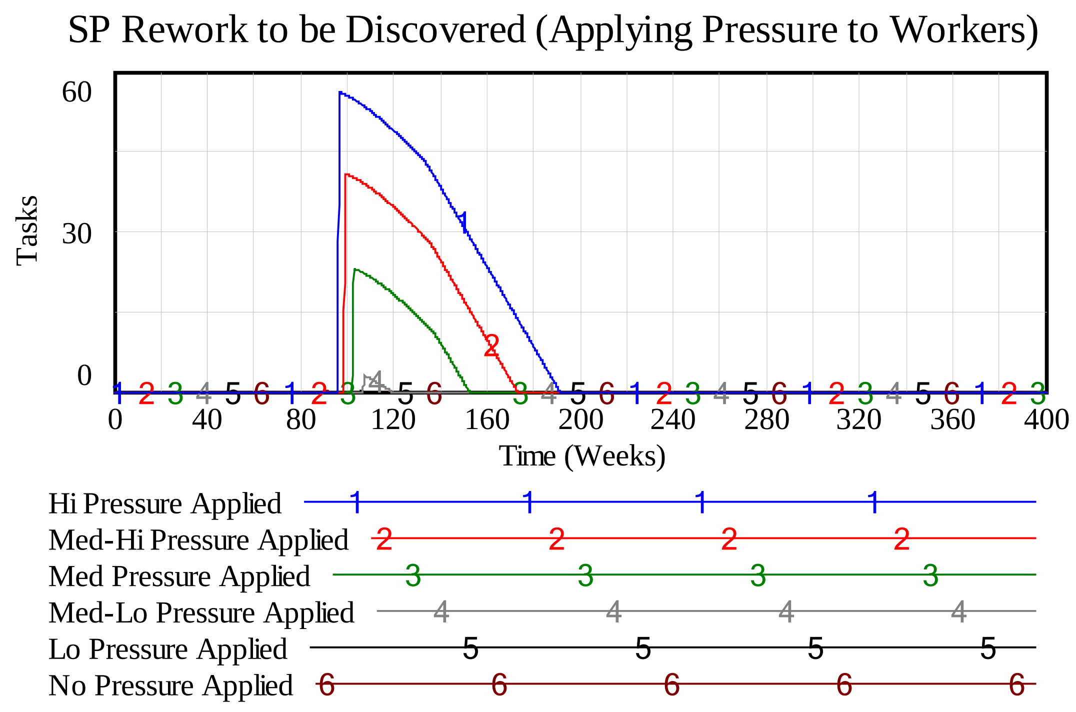


Figure 8: Simulation Results for Science Project Rework to be Discovered

Preliminary Finding 3: The tipping point contributes to the "90% Syndrome."

As described in Figure 2 of the causal hypothesis, the model simulation does reflect the 90% Syndrome. Surprisingly, as shown in Figure 9, the only place this phenomenon is seen is where the project has gone past the tipping point, i.e., where above moderate pressure is applied to workers to meet schedule demands in simulation runs 4, 5, and 6.[10] It is significant that the simulation's behavior in seeing forward progress plateauing well before project completion is very consistent with the predictions of the 90% Syndrome. The fact that the simulation plateaus near the 80% level may be due to the fact that the model takes into account all rework remaining in its calculations.

[10] The simulation runs for Percentage Complete at the lower pressure levels (runs 4, 5, and 6) shown in the figure do not actually show 100% completion in the second half of the simulation. The project does terminate, but some minimal level of rework is allowed to remain even at project conclusion—i.e., not all loose ends are completely tied up.

One unexpected observation of the above dynamic is that it appears the simulation requires going over the tipping point before the project displays the 90% syndrome behavior. Progress on the projects that do complete (i.e., simulation runs 4 through 6 in Figure 9) appears to increase in the final weeks. It isn't clear whether actual acquisition programs that exhibit the 90% syndrome are necessarily "over the tipping point" for that project. It may be unlikely, but if true it would be informative about what needs to be done in those circumstances (e.g., letting the project run its course without rebaselining—including schedule relief—might not be the wisest option, as it would never complete).

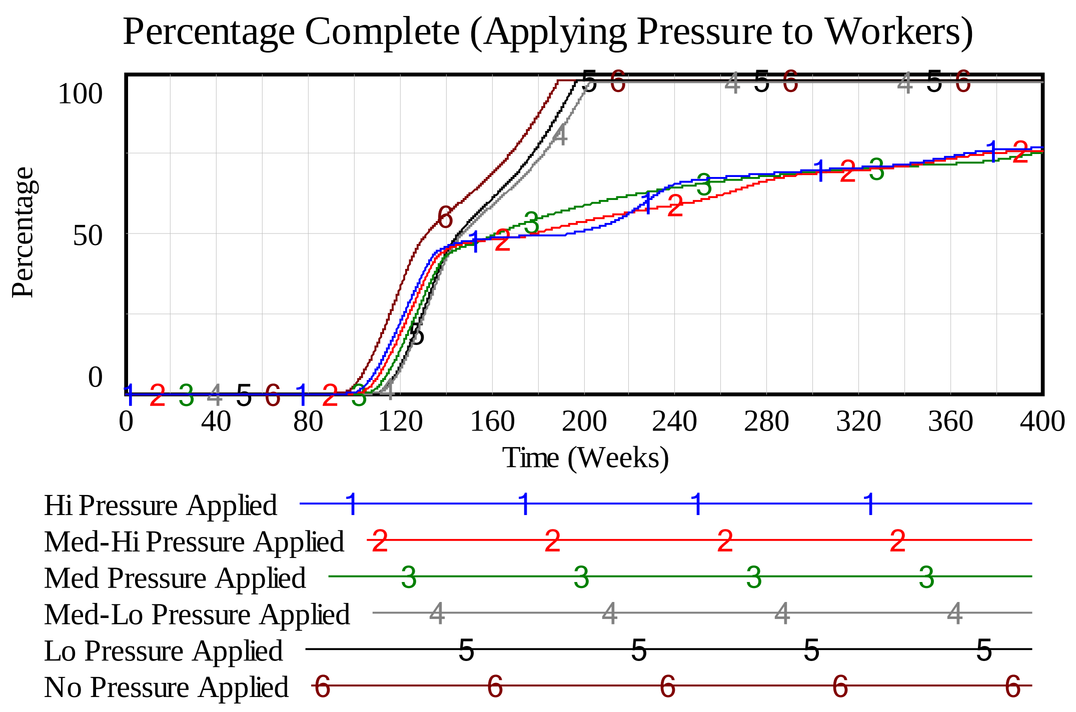


Figure 9: Simulation Results for Percentage Complete

Preliminary Finding 4: Applying only a low level of pressure to workers for a limited time, even in the face of heavy schedule demands, shortens project duration through increased productivity.

The above simulation graphs might seem to suggest that the less pressure applied to workers, the better. However, a VenSim optimization shows that a steady, low level of pressure is optimal for reducing project duration. While this has not been tested explicitly as part of the model, it may be that by allowing periods of heightened pressure on workers, followed by periods of relaxation, the program could limit the potential for worker burnout, and perform better regarding schedule. This may be a point of future investigation.

Preliminary Finding 5: From a technical standpoint, an earlier transition from the science project to production development creates fewer problems than a later transition does.

Figure 10 shows the accumulation of PD Discovered Quality Issues when the scope of the SP effort is varied. We measure the scope of the SP effort based on the number of features developed, but show it qualitatively in the graph on a four-point scale: Low, Medium-Low, Medium, or High. A medium SP scope was the assumed level in all previous simulations represented above. To simplify the analysis we assume that all schedule pressure is applied to the workers as the science project develops the prototype, so the simulation goes past the tipping point in the case of a medium-feature scope. As is apparent from the figure, going to a high-feature scope (in simulation run 2) causes a later start of the production development, again surpasses the tipping point, and results in higher amplitude oscillations. Reducing the scope to medium-low (in run 3) is still beyond the tipping point, but results in a significantly damped oscillation. Finally, reducing the scope of the development to a low level (simulation run 4) results in successful completion of the production development at approximately week 140, without the project passing the tipping point. The only difference between this run which successfully completed and the runs that passed the tipping point was the scope of the science project prototype. Clearly a science project prototype needs to demonstrate viability and utility, but this simulation shows that going beyond that can lead to disastrous consequences for the project as a whole.[11]

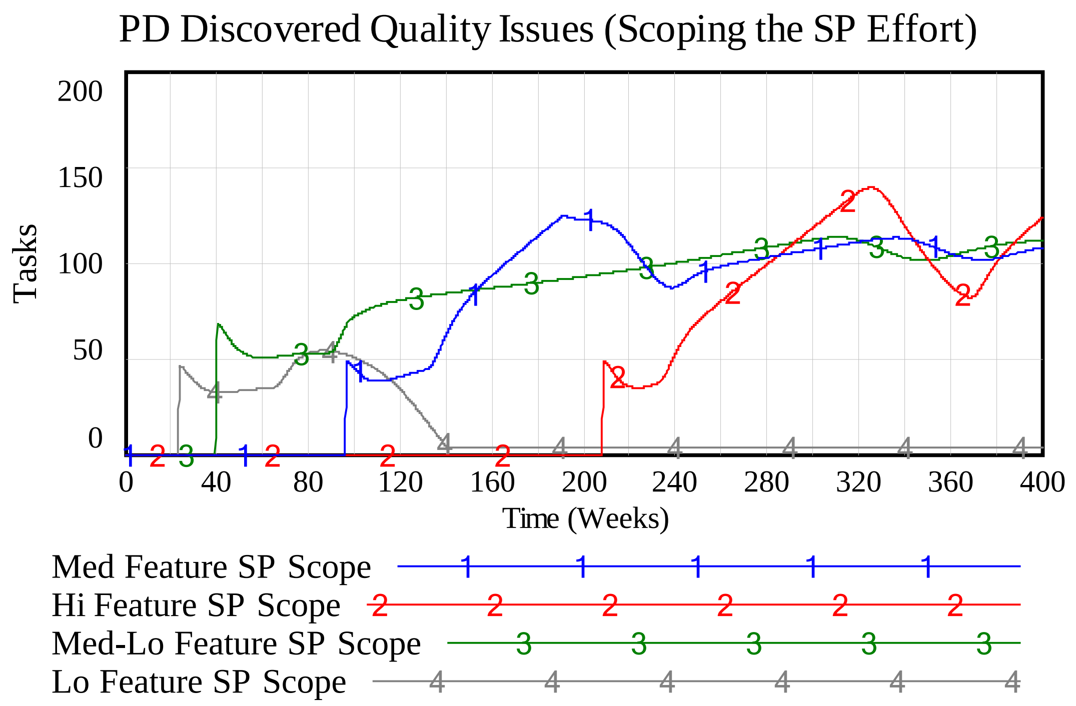


Figure 10: Simulation Results for Production Development Discovered Quality Issues

Preliminary Finding 6: The project performs significantly better by shelving the prototype when it is complete, instead of reusing a poorly-constructed prototype

Not unexpectedly, the simulation predicts a better outcome for shelving the prototype before moving to production development. It is difficult to envision a worse outcome than a project that goes over the tipping point, spiraling downward towards more and more rework. Fortunately, shelving the prototype and starting over in production development does not also go over the tipping point—it was the decision to reuse the poorly-constructed prototype that caused the failure. Figure 11 shows that not reusing the prototype in simulation run 2 leads to project completion around week 180, whereas reusing the prototype leads to ever increasing estimated completion dates. The primary reason is the reduction in the amount of rework the production development has to deal with as shown in Figure 12.

[11] Appendix H provides other simulation behaviors for various levels of prototype scoping.

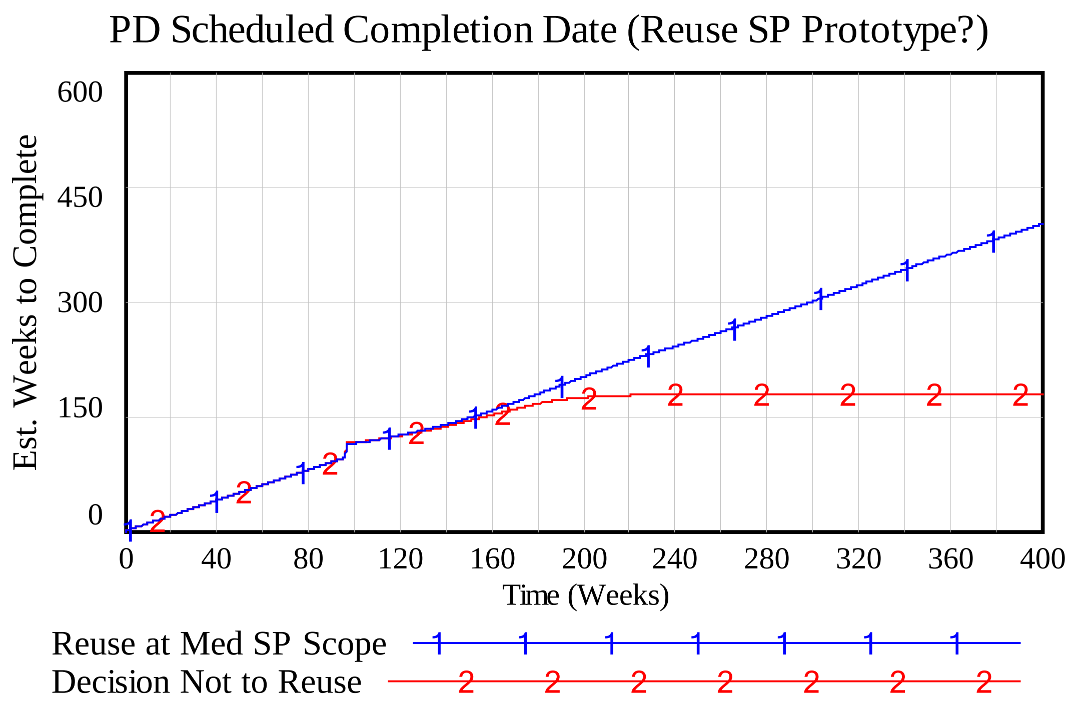


Figure 11: Simulation Results for Production Development Scheduled Completion Date Reflecting the Decision to Reuse

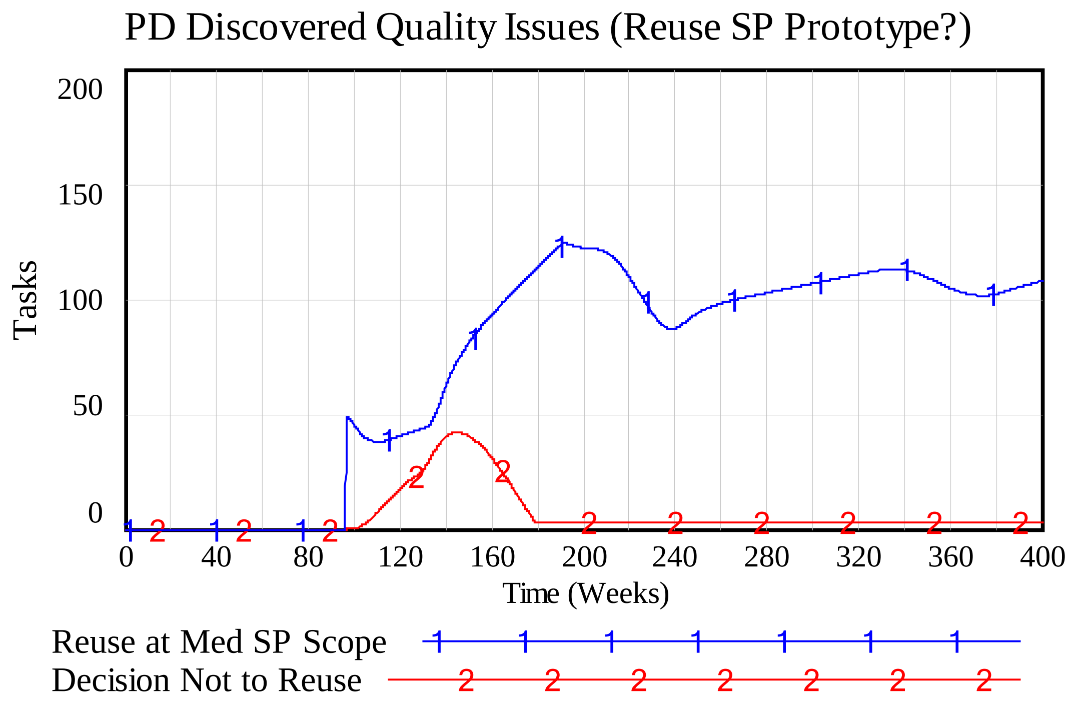


Figure 12: Simulation Results for Production Development Discovered Quality Issues Reflecting the Decision to Reuse

That said, it was not strictly the decision to reuse the prototype that created the disastrous results. Figure 13 below shows that reusing better-constructed prototypes results in approximately the same project duration. In other words, the benefits of doing a science project can be achieved while still keeping to a reasonable schedule, as long as the quality of the prototype does not slip. Interestingly, the best project performance observed was where the science project prototype scope was low, in which case the production development reused the prototype to finish the project in under 150 weeks.

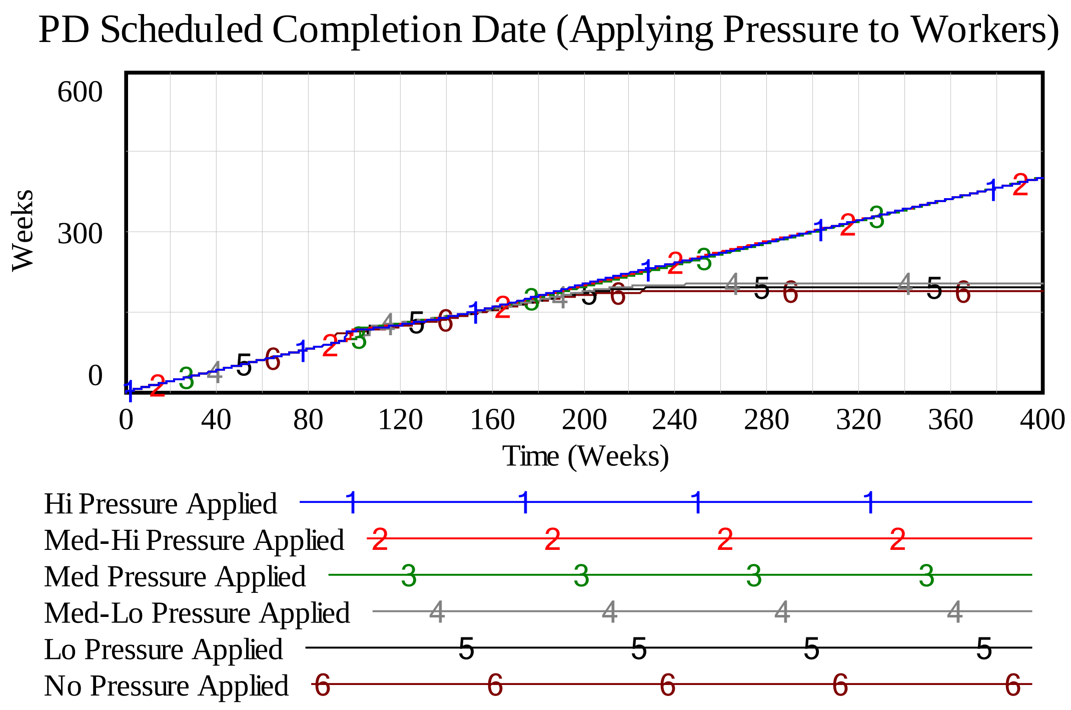


Figure 13: Simulation Results for Production Development Scheduled Completion Date for Various Worker Pressures

A key conclusion from this investigation is that the adverse consequences of “The Evolution of a Science Project” dynamic will take hold if the project develops a “throwaway” prototype and nonetheless attempts to evolve it into a production-quality system. The project will likely also slip schedule significantly if it stops development, discards the “throwaway prototype,” and redesigns/re-implements the prototype from the beginning (perhaps incorporating the lessons learned from the prototype implementation)—but it will likely still complete development of the system. If those were the only two options, the question would be one of when to discard the prototype and shift over to conducting a formal acquisition program. A third possible option may be to deploy the last viable version of the prototype, stop development on the prototype, stop using it for further new development, and use the final prototype release as a way to “buy some time” with users—while the development team begins the formal reimplementation effort. However, this approach also has a drawback in terms of the prototype maintenance; when the prototype system does fail in a significant way in the field, someone will likely have to be assigned to understand what went wrong, and how to work around the problem, consuming more of the limited project resources.

However, experience suggests that the path that is most likely to be most successful in a science project context is to develop an *evolutionary* prototype so that the project will neither “hit the wall” when development stalls, nor “go dark” when the prototype must be discarded and re-implemented. Because there is less build-up of undiscovered rework, and no time-consuming restart and re-implementation, development can continue through to completion at a reasonable pace. The problem, though, originates with the fundamental conditions and incentives that tend to drive the creation of science projects. If the development effort is led by operational, functional, or business people, their lack of software expertise may make them unaware of this approach, and thus preclude them from using it. Even if the evolutionary prototyping approach were used, it would likely also cost more, and delay the system's first deployment to the field. Neither the time to develop a quality system architecture and design, or produce comprehensive documentation, will be small.

5 Conclusion

Software acquisition programs regularly experience recurring cost, schedule, and quality failures, and progress and outcomes often appear to be unpredictable and unmanageable. The subject of this paper is a specific adverse acquisition dynamic concerning the poorly controlled evolution of small prototype efforts into full-scale systems that is called "The Evolution of a Science Project." This paper qualitatively describes key aspects of the dynamic, describes a system dynamics model that was built to analyze the dynamic, and presents six preliminary findings:

- The accumulation of undiscovered rework from the science project creates a tipping point during production development (*at the tipping point program behavior changes unexpectedly from completing in a finite time, to not completing at all*).
- Applying high pressure to developers, or even moderate pressure for extended periods, can push the program past the tipping point.
- The tipping point contributes to the 90% Syndrome (this occurs when development stalls near completion, with little progress made despite spending more time and effort).
- Applying only a low level of pressure to workers for a limited time, even in the face of heavy schedule demands, shortens project duration through increased productivity.
- From a technical standpoint, an earlier transition from the science project to production development creates fewer problems than a later transition does.
- The project performs significantly better by shelving the prototype when it is complete, instead of reusing a poorly constructed prototype.

The full report on our effort describes general lessons learned about the system dynamics modeling process and tools (Novak, Moore, & Alberts, 2012). It also makes six observations about the activity of system dynamics modeling of software-reliant acquisition that are drawn from our experiences both with this effort, and with software-reliant acquisition programs in general:

- Many software-reliant acquisition program behaviors are an interaction between the inherent structural dynamics (i.e., the "physics") of the software development activities, and the incentives that are acting upon the key participants and stakeholders as they make decisions.
- System dynamics modeling can provide a deep understanding of the often poorly understood dynamics of software-reliant acquisition.
- It's helpful to have a clear understanding of a given dynamic before selecting the most appropriate course of action for a given program (so that the action addresses the actual problem, rather than only a symptom).
- Tipping points may play a key role in many complex and challenging software development and acquisition situations, and require further study.
- Building models that are valid and reliable is a significant undertaking, requiring significant effort, as well as both detailed domain expertise and extensive modeling experience.
- There is much more that can be learned about the problematic behaviors of software-reliant acquisition programs and their potential solutions by applying system dynamics modeling to other aspects of software-reliant acquisition.

Beyond extending the current model in several areas where we see it deficient (see Appendix J of (Novak, Moore, & Alberts, 2012)), future work would involve using the model to

- *Model potential mitigations and solutions* - An important next step in this work is to mitigate the effects of misaligned acquisition program organizational incentives and adverse software-reliant acquisition structural dynamics by improving program staff decision-making. This requires not only modeling and analyzing adverse acquisition dynamics that have been encountered in actual programs, but identifying and then modeling candidate solutions that may be able to mitigate or resolve the counter-productive behaviors. Constructing potential solutions and then connecting them to the original model of the program behavior makes it possible to evaluate the ability of each candidate approach to mitigate the dynamic's adverse behavior. This allows an initial assessment of the value of the approach to be made, which could then be initially piloted on a small program to see how well the predicted model behavior mapped to the real world.

- *Teach important model-based lessons learned using experiential learning* - Education in a classroom setting does not always translate to the reality of complex challenges and changing priorities students face in an operational environment—and as a result, may not produce significant job performance improvement. System dynamics models provide an opportunity to provide experiential learning by allowing them to interact with the dynamics of the kind of problems they may face in an operational environment. The system dynamics model developed for "The Evolution of a Science Project" dynamic can provide the foundation for an interactive learning simulation. Using interactive simulations to learn about acquisition helps significantly in both analyzing and making decisions about these complex situations. By improving program decision-making, we can help programs overcome counter-productive behaviors that stem from misaligned incentives, and thus deploy higher-quality systems to the field in a more timely and cost-effective manner.

In conclusion, we believe that *modeling the incentives and dynamic behavior of acquisition programs is one of the most powerful and cost-effective ways available of understanding problems, evaluating candidate solutions, and producing better acquisition program outcomes*. It is neither cost-effective nor practical to conduct large-scale experiments on acquisition programs of record, which makes modeling these systems the most viable method for studying them and analyzing the comparative advantages of different practices and methods. Even the most minor efficiencies gained in large acquisition programs from using modeling could produce benefits in functionality, quality, performance, improved time-to-deployment, and direct cost savings that would recoup the research and analysis investment many times over. We hope to continue to have the opportunity in future work to prove this to be the case.

6 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. We would also like to thank the SEI's Acquisition Support Program Director Mary Catherine Ward for her support and encouragement. Finally, the authors owe a debt of gratitude to the people within the SEI who provided technical review and comment on the report on which this paper is based, including John Foreman, Christopher Alberts, Julie Cohen, and Jay Marchetti, as well as our editor, Gerald Miller.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

References

Abdel-Hamid, T., & Madnick, S. (1991). *Software Project Dynamics: An Integrated Approach.* Prentice Hall.

Abelson, L. A., Adams, R. J., & Eslinger, S. (2004, January). Acquisition Modeling: The Kkey to Managing Acquisition Complexity. *Proceedings of the Third Annual Conference on Software-Intensive Systems Acquisition.*

Birkholzer, T., Madachy, R. J., Pfahl, D., Beitinger, H., Schuster, M., & Olkov, A. (2010). SimSWE - A Library of Reusable Components for Software Process Simulation. *ICSP,* (pp. 321-332).

Brougham, W. (1999). *System Dynamics and Process Improvement: Can the U.S. Navy Acquisition Community Learn from Industry Behavior?* Center for Information Systems Research. Cambridge: Alfred P. Sloan School of Managment.

Buettner, D. J. (2008). *Designing an Optimal Softwware Intensive system Acquisition: A Game Theoretic Approach.* University of Southern California.

Choi, K., & Bae, D.-H. (2006). Analysis of Software-Intensive System Acquisition Using Hybrid Software Process Simulation. In *Sooftware Process Change, Lecture Notes in Computer Science* (Vol. 3966, pp. 254-261). Springer-Verlag.

Cooper, K. G. (1980, December). Naval Ship Production: A Claim Settled and a Framework Built. *Interfaces, 10*(6), 20-36.

Ford, D. N., & Sterman, J. D. (1998). Dynamic Modeling of Product Development Processes. *System Dynamics Review, 14*(1), 31-68.

Haberlein, T. (2004, September). Common Structures in System Dynamics Models of Software Acquisition Projects. *Softare Process Improvement and Practice*, 67-80.

Kadish, R. (2006). *Defense Acquisition Performance Assessment Report - Assessment Panel of the Defense Acquisition Performance Assesment Project.* Defense Acquisition University.

Li, J., Taylor, T., & Ford, D. N. (2011). Impacts of Project Controls on Tipping Point Dynamics in Construction Projects. *Proceedings of the 29th Annual International Conference of the System Dynamics Society*. Washington, D.C.

Li, Y. (2011). The Impact of Design Rework on Construction Project Performance. *Proceedings of the 29th Annual International Conference of the System Dynamics Society.* Washington, D.C.

Lyneis, J. M., & Ford, D. N. (2007). System Dynamics Applied to Project Management: A Survey, Asssessment, and Directions for Future Research. *System Dynamics Review, 23*(2/3).

Madachy, R. J. (2008). *Software Process Dynamics.* Wiley-IEEE Press.

Madachy, R., Boehm, B., & Lane, J. (2006, May). Spiral Lifecycle Increment Modeling for New Hybrid Processes. *Proceedings of SPW/ProSim'2006*, (pp. 167-177). Shanghai.

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

Novak, W. E., Moore, A. P., & Alberts, C. (2012). *The Evolution of a Science Project: A Preliminary System Dynamics Model of a Recurring Software-Reliant Acquisition Behavior.* Carnegie Mellon University. Pittsburgh: Software Engineering Institute.

Raffo, D. M., & Kellner, M. I. (2000). Empirical Analysis in Software Process Simulation Modeling. *Journal of Systems and Software, 47*(9).

Repenning, N. P., Goncalves, P., & Black, L. J. (2001, July). Past the Tipping Point: The Persistence of Firefighting in Product Development. *Californial Management Review*.

Scacchi, W., & Choi, J. S. (2001, December). Modeling and Simulating Software Acquisition Process Architectures. *Journal of Systems and Software, 59*(3), 343-354.

Scacchi, W., & Mi, P. (1993). Modeling, Enacting, and Integrating Software Engineering Processes. *Proceedings of the 3rd Irvine Software Symposium.* Costa Mesa (California).

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

Taylor, T., Ford, D. N., & Johnson, S. (2005). Why Good Projects Go Bad: Managing Development Projects Near Tipping Points. *Proceedings of the 23rd International Conference of the System Dynamics Society*. Boston.

U.S. GAO. (2005). *DoD Acquisition Outcomes: A Case for Change - Statement of Katherine V. Schiasi, Managing Director.* United States Government Accountability Office.

Wirthlin, J., Houston, D., & Madachy, R. (2011). Defense Acquisition System Simulation Studies. *Proceedings of the International Conference on Software and System Process.* Waikiki.

Appendix A: System Dynamics Background

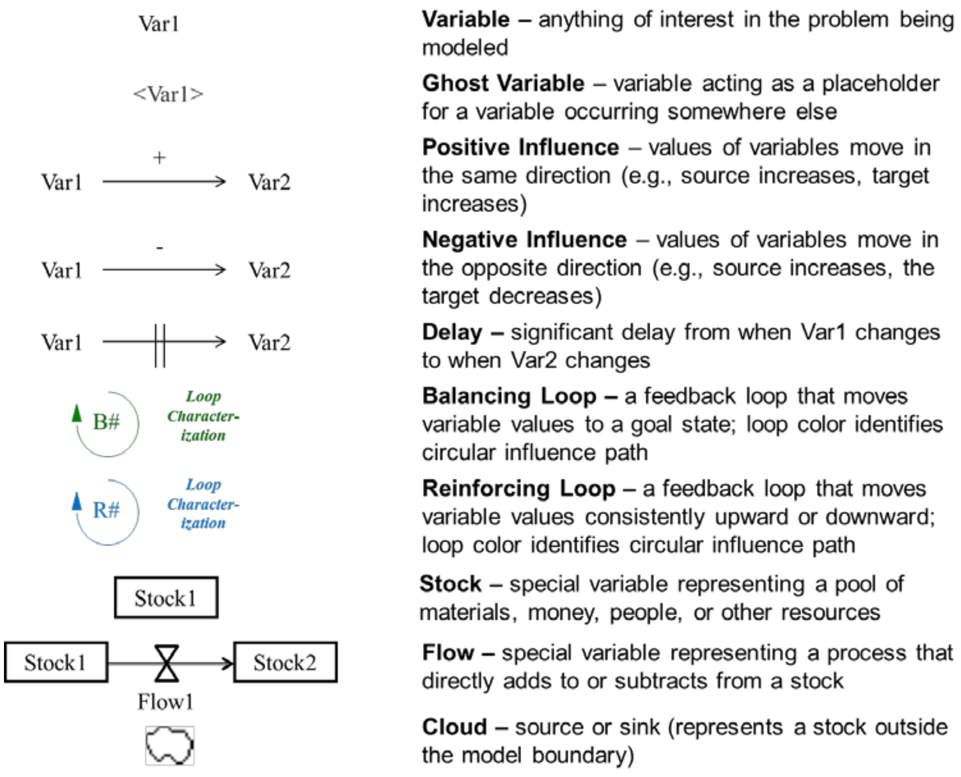
The system dynamics method helps analysts model and analyze critical behavior as it evolves over time within complex socio-technical domains. A powerful tenet of this method is that the dynamic complexity of critical behavior can be captured by the underlying feedback structure of that behavior. The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

- Sterman’s comprehensive treatment of the system dynamics method (Sterman, 2000)
- Meadows’ description of thinking in systems rather than components in isolation as a means to analyze and solve problems (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, or cultural factors. The exclusion of soft factors in other modeling techniques essentially 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 14 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. Basically, a positive (+) influence indicates that the values of the variables move in the same direction, whereas a negative (-) 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. 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.

System dynamics models identify two types of feedback loops: balancing and reinforcing. Significant feedback loops identified within a model are indicated by a loop symbol and a loop name in italics. Balancing loops—indicated with the label *B* followed by a number in the loop symbol—describe aspects of the system that oppose change, seeking to drive variables to some goal state. Balancing loops often represent actions that an organization takes to 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 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

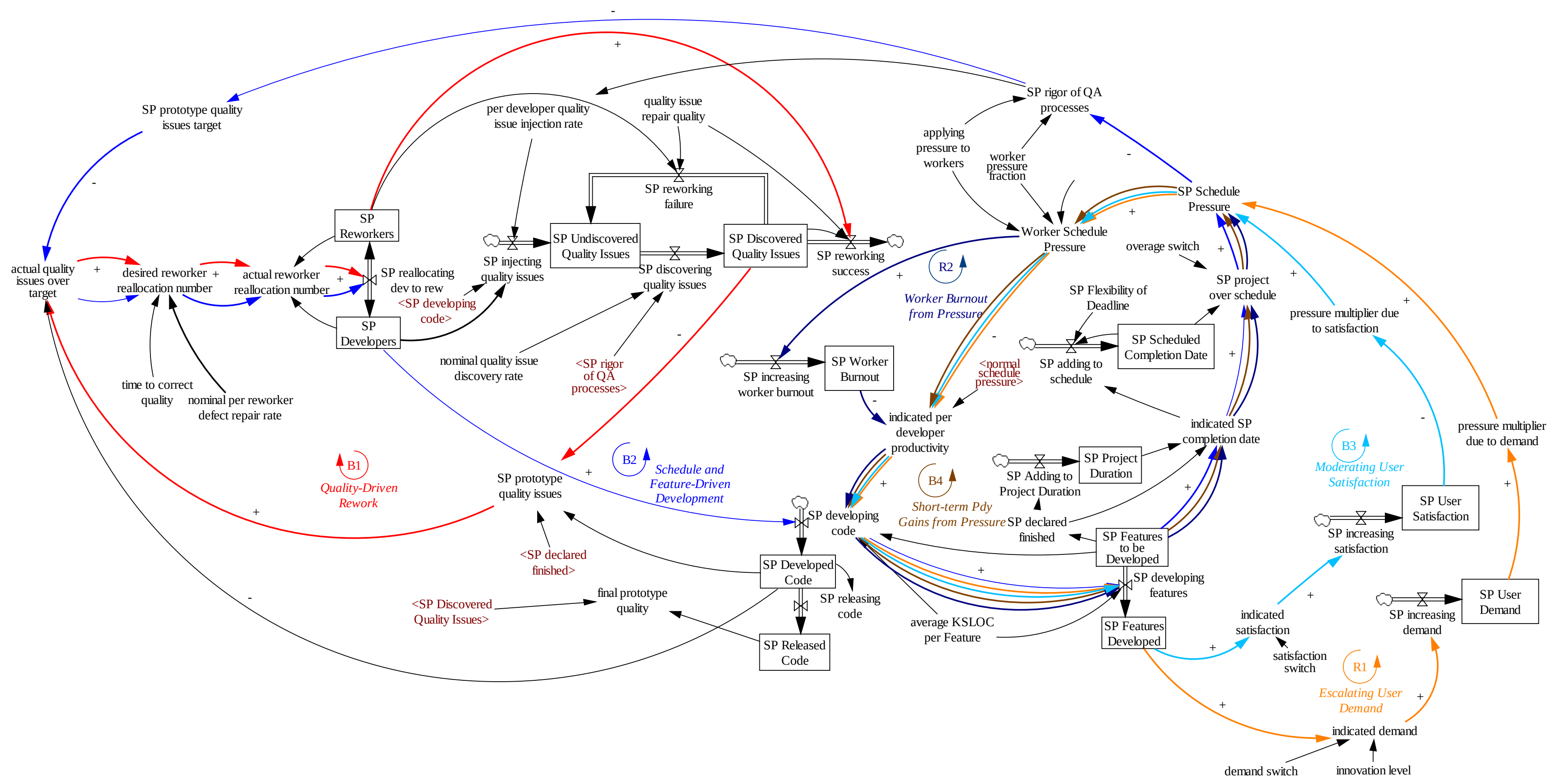
Stock – special variable representing a pool of materials, money, people, or other resources

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)

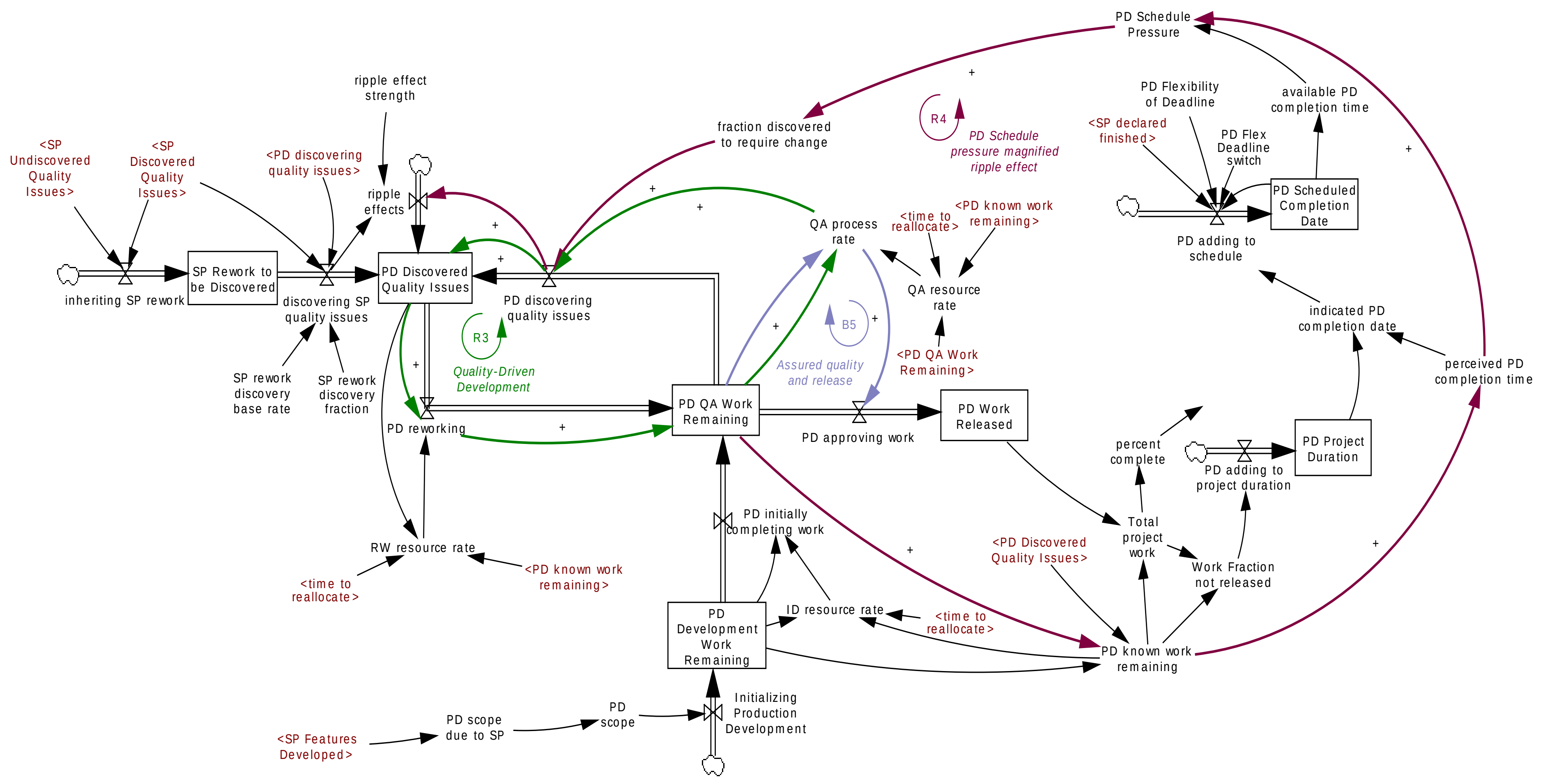
Figure 14: System Dynamics Notation

Appendix B: System Dynamics Simulation Model: Science Project Sector



SP prototype quality issues target
actual quality issues over target
desired reworker reallocation number
actual reworker reallocation number
SP Reworkers
SP Developers
SP reallocating dev to rew
<SP developing code>
time to correct quality
nominal per reworker defect repair rate
per developer quality issue injection rate
quality issue repair quality
SP reworking failure
SP injecting quality issues
SP Undiscovered Quality Issues
SP discovering quality issues
SP Discovered Quality Issues
SP reworking success
nominal quality issue discovery rate
<SP rigor of QA processes>
B1
Quality-Driven Rework
B2
Schedule and Feature-Driven Development
SP prototype quality issues
<SP declared finished>
<SP Discovered Quality Issues>
final prototype quality
SP Developed Code
SP releasing code
SP Released Code
SP developing code
SP rigor of QA processes
applying pressure to workers
worker pressure fraction
Worker Schedule Pressure
SP Schedule Pressure
R2
Worker Burnout from Pressure
SP increasing worker burnout
SP Worker Burnout
indicated per developer productivity
<normal schedule pressure>
overage switch
SP project over schedule
SP Flexibility of Deadline
SP adding to schedule
SP Scheduled Completion Date
indicated SP completion date
pressure multiplier due to satisfaction
B4
Short-term Pdy Gains from Pressure
SP Adding to Project Duration
SP Project Duration
SP declared finished
SP Features to be Developed
SP developing features
average KSLOC per Feature
SP Features Developed
B3
Moderating User Satisfaction
SP User Satisfaction
SP increasing satisfaction
indicated satisfaction
satisfaction switch
pressure multiplier due to demand
SP User Demand
SP increasing demand
R1
Escalating User Demand
indicated demand
demand switch
innovation level

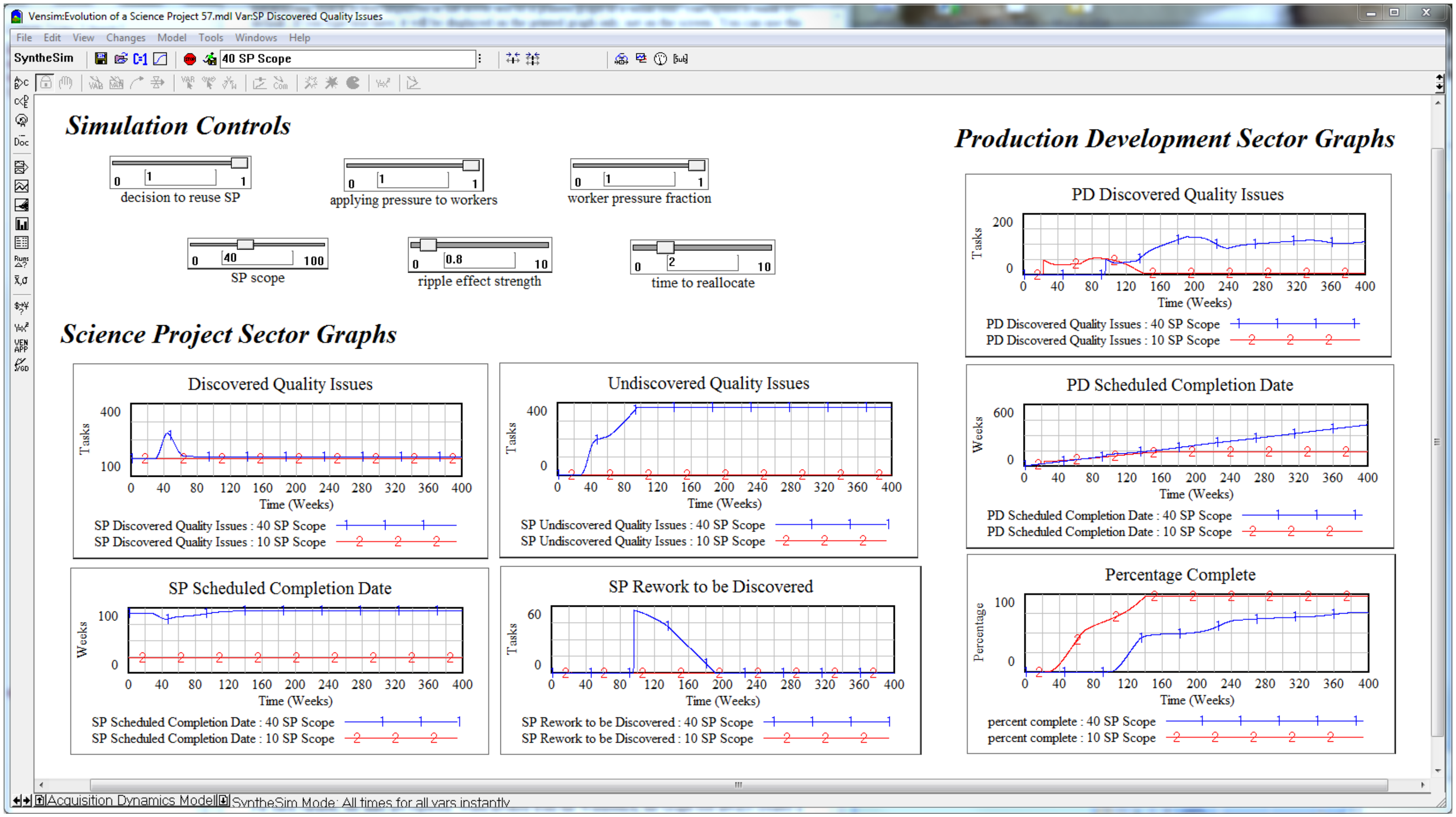
Appendix C: System Dynamics Simulation Model: Production Development Sector



PD Schedule Pressure
ripple effect strength
fraction discovered to require change
R4
PD Schedule pressure magnified ripple effect
PD Flexibility of Deadline
available PD completion time
<SP declared finished>
PD Flex Deadline switch
<SP Undiscovered Quality Issues>
<SP Discovered Quality Issues>
<PD discovering quality issues>
ripple effects
QA process rate
<time to reallocate>
<PD known work remaining>
PD Scheduled Completion Date
PD adding to schedule
SP Rework to be Discovered
inheriting SP rework
discovering SP quality issues
PD Discovered Quality Issues
PD discovering quality issues
QA resource rate
R3
Quality-Driven Development
B5
Assured quality and release
indicated PD completion date
SP rework discovery base rate
SP rework discovery fraction
<PD QA Work Remaining>
perceived PD completion time
PD QA Work Remaining
PD Work Released
PD reworking
PD approving work
percent complete
PD adding to project duration
PD Project Duration
PD initially completing work
Total project work
RW resource rate
<PD Discovered Quality Issues>
Work Fraction not released
<time to reallocate>
<PD known work remaining>
PD Development Work Remaining
ID resource rate
<time to reallocate>
PD known work remaining
PD scope
PD scope due to SP
Initializing Production Development
<SP Features Developed>

Appendix D: Interface for Interacting with the Model

Note that in this user interface, there are two user controls provided to change the amount of schedule pressure that is being applied to developers: applying pressure to workers, and worker pressure fraction. Although it's not discernible in the screen capture, the control for applying pressure to workers is binary, either allowing the pressure, or not. The control for worker pressure fraction is a real number, and varies continuously between 0 and 1 to define the precise level of that pressure—but is only active if applying pressure to workers is also set to 1 (i.e., "true").


Vensim:Evolution of a Science Project 57.mdl Var:SP Discovered Quality Issues

SyntheSim 40 SP Scope

Simulation Controls

decision to reuse SP: 1 (0–1)
applying pressure to workers: 1 (0–1)
worker pressure fraction: 1 (0–1)
SP scope: 40 (0–100)
ripple effect strength: 0.8 (0–10)
time to reallocate: 2 (0–10)

Acquisition Dynamics Model SyntheSim Mode: All times for all vars instantly

File Edit View Changes Model Tools Windows Help
Science Project Sector Graphs
Production Development Sector Graphs
Discovered Quality Issues
Tasks
Time (Weeks)
SP Discovered Quality Issues : 40 SP Scope
SP Discovered Quality Issues : 10 SP Scope
Undiscovered Quality Issues
SP Undiscovered Quality Issues : 40 SP Scope
SP Undiscovered Quality Issues : 10 SP Scope
SP Scheduled Completion Date
Weeks
SP Scheduled Completion Date : 40 SP Scope
SP Scheduled Completion Date : 10 SP Scope
SP Rework to be Discovered
SP Rework to be Discovered : 40 SP Scope
SP Rework to be Discovered : 10 SP Scope
PD Discovered Quality Issues
PD Discovered Quality Issues : 40 SP Scope
PD Discovered Quality Issues : 10 SP Scope
PD Scheduled Completion Date
PD Scheduled Completion Date : 40 SP Scope
PD Scheduled Completion Date : 10 SP Scope
Percentage Complete
Percentage
percent complete : 40 SP Scope
percent complete : 10 SP Scope