Acquisition Archetypes

The Hidden Laws of Software-Intensive Development Programs

William E. Novak, SEI
Andrew P. Moore, SEI

Abstract. Many of the behaviors and adverse outcomes that we see in software-intensive programs are the result of "misaligned incentives" between the goals of the individuals involved and those of the larger organization. These interact and play out in recurring dynamics that are familiar to both software developers and managers, but which are still poorly understood. By characterizing the forces within these dynamics explicitly in the form of the "acquisition archetypes" described in this paper we can come to understand the underlying mechanisms that cause these problems, and identify mitigations to help mitigate and prevent them.

Introduction

Software development, especially in the context of defense acquisition programs, displays a set of all-too-familiar outcomes that seem to point to a set of common causes. We see these repeatedly in programs: making up schedule delays by cutting corners on quality activities, postponing risky development tasks until later development spirals, failing to identify critical risks to senior management, underestimating cost by large margins, and many others. We know these patterns and outcomes occur; what we have difficulty understanding is the mechanism which causes them.

The SEI regularly engages with acquisition programs by conducting in-depth Independent Technical Assessments to assess program status, understand the reasons behind specific challenges, and make recommendations for corrective actions and future prevention. These assessments examine different aspects of programs, combining document review and code analysis with face-to-face interviews of program, contractor, and other stakeholder staff. The analyses that have been conducted expose many of the forces that drive these programs, and have provided a detailed portrait of some of the most common pitfalls that programs face.

If we wonder why some of these problems continue to occur, we must realize that it can be difficult to recognize the patterns of the problems that surround us simply because we are standing too close to them. If we do not see the larger patterns that they belong to, we will likely fail to recognize even familiar problems when we encounter them in new circumstances.

This paper explains an approach to thinking about recurring acquisition problems, and presents several examples of "acquisition archetypes" that characterize the structure of the forces that drive various counter-productive software-intensive acquisition program behaviors. These archetypes represent a set of omnipresent, and yet frequently ignored, "laws" of software development. Although ubiquitous, there are ways to get around these laws—and approaches to both mitigating and preventing these behaviors, based on the understanding of the underlying structure, are discussed.

Complex, Dynamic Systems and Acquisition Programs

Our focus in large-scale software development is commonly on the complexity and challenges offered by the system that is being developed. However, one of the reasons that successfully completing a software-intensive acquisition effort can be so hard is that these programs themselves are complex, dynamic systems. They feature complex interactions between the PMO, contractors, subcontractors, test organizations, sustainment organizations, sponsors, and users—all of whom act largely autonomously, and in their own interests. There is limited visibility into actual program progress and status. There are often significant delays between making changes to the system, and seeing their results, making the link between cause and effect within the system unclear. There is feedback that occurs between the decisions and actions of the different stakeholder entities, causing seemingly unpredictable results. The feedback can then produce situations that can escalate despite management's best efforts to control them.

These types of systems can trap people into certain behaviors that are ultimately driven by the system. As a simple example, we can think of the stock market, where people tend to buy when the market is bullish, and sell when it is falling. There is no intention on the part of individuals to cause or contribute to the creation of a market "bubble" or a market crash—and yet that is precisely what our collective behaviors do, even though we are only acting in our own self-interest. Our actions in the context of a complex, dynamic system often have unintended consequences which can make things worse. We are trapped by the ways our rational decisions (as they may appear to us to be) interact with the dynamics of the larger system to which they belong.

Misaligned Incentives and Social Dilemmas

There are incentives within most organizations that work at cross purposes with one another—which are "misaligned"—in that they do not combine to cause actions that produce the desired result. This misalignment can result in ineffective decision-making in which short-term interests take precedent over more strategic longer-term interests, or the objectives of the larger organization can take a back seat to individual or team goals. We may be inclined to think that the recurring behaviors we see in organizations are simply the result of individual personalities and their different styles. While these differences may have significant effects, they cannot explain the recurring nature of these behaviors—and so they are not as important as the contextual structure of laws, regulations, rules, guidelines, and preferences in which people operate. As Peter Senge observed, "When placed in the same system, people, however different, tend to produce similar results." Economists believe that people respond to incentives, and this is correct. We should not expect to rely upon the integrity of people to achieve an organization's goals if the organization's policies and incentives oppose them. While people want to do "the

right thing," when they are forced to choose between personal self-interest and organizational goals, the temptation toward self-interest can be too great.

Some examples of misaligned incentives that occur in software-intensive acquisition include:

- The preference for using the most advanced technology, even if it may be immature. The government wishes to provide the most powerful capability to the warfighter, and the contractor prefers to enhance their experience base with the latest technologies—even if the risk of using them is higher.
- The preference for longer duration programs, which allow the government to build greater capability systems, and offer contractors greater staff and revenue stability. However, they increase the risk of scope creep from advancing technology during development.

Many misaligned incentives can be classified as what sociologists and others call "social dilemmas." Social dilemmas describe situations in which the most likely solution to spontaneously emerge is one that may be optimal for the individuals involved, but will likely be suboptimal overall.

One of the most common types of social dilemmas is the social trap. A social trap is a situation where an individual desires a benefit (often by exploiting a shared resource) that will cost everyone else—but if all in the group succumb to that same temptation, then everyone will be worse off, because the common resource will eventually be depleted.

A social trap is often referred to as a "Tragedy of the Commons[1]." The interesting thing about a social trap is that the people involved do not intend to harm themselves or others by their decisions—they are all simply acting in their own self interest—but the "tragic" collective result of depleting the resource is still almost inevitable.

Social traps are not rare—we see examples of them every day: overfishing, traffic congestion, and air and water pollution are all the results of large-scale social traps. These are the unintended consequences (i.e., what economists call "externalities") of our intended activities: catching fish to eat, travelling to other places for work and pleasure, and producing goods and services that we need. In these social dilemma situations, to paraphrase economist Adam Smith, "Individually optimal decisions lead to collectively inferior solutions." Furthermore, because they can appear in so many different forms, they are difficult both to recognize and to fix.

We see an instance of a social trap in joint acquisition programs that attempt to build a single capability that will be used by multiple stakeholders. As more stakeholders agree to participate, they each bring new, unique needs to the joint program office (JPO). If the JPO rejects these additional requirements, they risk driving the stakeholders away, as the stakeholders would generally prefer to build a custom system. However, if the JPO accepts the requirements to satisfy the stakeholders, then doing so will likely drive up the cost, schedule, risk, and complexity of the joint program—and drive the stakeholders away for different reasons.

Systems Thinking

One tool for analyzing complex, dynamic systems is systems thinking—a method that uses the identification of feedback loops to analyze common system structures that either regulate themselves, or may escalate or decline. Systems thinking has its roots in system dynamics work pioneered by Jay W. Forrester at MIT in the 1960s, and views systems as sets of components with complex interrelations occurring between them. A widely used tool for systems thinking is the causal loop diagram, which explicitly represents the feedback loops in the system, showing the driving forces, or causes, of the overall system behavior.

The value of systems thinking is that such diagrams can help to identify the underlying structure of a system, which is what drives the behavior that we see. This is important because without an understanding of that structure, applying solutions to address problems in complex, dynamic systems may have unexpected side-effects that can make things worse. Lasting improvement for such systems may only come from changing the underlying system structure.

One tenet of systems thinking is that the behavior of a system is greater than the aggregate of its individual component behaviors. This "new" system behavior that results, which is generally not an intended result of the system, is called an emergent behavior. Emergent behaviors come about as the result of the interactions among the various rules (physical, legal, social, etc.) that govern the system. Examples of emergent behaviors include the ebb and flow of traffic, the flocking of birds, the meandering courses of rivers, the evolving patterns of cities and suburbs, the synchronized applause of enthusiastic audiences, market "crashes" or sell-offs, and many others. For our purposes here, the unintended consequences seen in systems thinking, both from interacting physical laws, and from the interactions of laws, regulations, policies, guidelines, preferences, and our own decisions and actions, are emergent behaviors.

Software Project Management

Clearly large software development programs are themselves complex, dynamic systems—which may be as complex, or more complex, than the software systems they are developing. Because of their increasing size and complexity, as evidenced by their inconsistent performance and outcomes, our projects may already be growing past the ability of our present management techniques to effectively manage them. While we focus much of our attention on the technical software systems that are our primary goal, we may ignore the fact that software development projects and programs feature autonomous, adaptive elements called "software developers" and "software managers" whose complexity is still poorly understood. The claim has been made by many experts that technical and software engineering issues may not be the primary reasons that many development programs fail. The main culprit in poor program outcomes may be the interactions of the people in the development organizations. Realizing this, as engineers we may look to the technology we understand best as a way to correct and prevent these problems—but as Edward's Law states, "Sociological problems do not always have technological solutions." We may need to look elsewhere to resolve these issues.

In trying to understand how such problems with software development can occur, we need only consider that people who develop systems have incentives to "sell" them with optimistic claims of both substantial benefits and low cost. The prospective customer, who is looking for—and in most cases demanding—a

system with significant capabilities at low cost, is often all too willing to believe that this is possible. The combination of incentives amounts to a "conspiracy of hope" among the stakeholders that all will turn out well, when in fact the opposite is more likely.

Program managers will generally focus on maximizing objectives where their performance is measured, and for which they will be rewarded (or penalized if they fail). Ancillary goals that are only desirable for the organization, and do not carry a personal incentive for being met, are unlikely to be achieved. Trying to ensure that a task will be performed by mandating PMs to do it just adds one more thing to an already overloaded plate—it provides only one incentive to do it, leaves in place both competing incentives as well as disincentives for ignoring it, and may not guarantee the quality with which it will be done (especially if the task is viewed as a "check the box" requirement). As an example, planning for software sustainment, while important, rarely has any bonus or penalty attached to it, is not a key performance metric for oversight like earned value management, and the quality of the planning work cannot easily be verified. Thus, the reduction of lifecycle costs through mandated sustainment planning is unlikely to be achieved.

When project managers are rewarded specifically for achieving certain goals, they will likely work hard to make those happen, even if that achievement must occur at the expense of other goals of the organization. There is rarely an incentive for an individual to make sacrifices for the common organizational good—which is, in part, why advancing it is so difficult to achieve.

Acquisition Archetypes

Acquisition archetypes are an adaptation and extension of Peter Senge's system archetypes work. The system archetypes each describe a recurring pattern of dynamic behavior that occurs in complex systems:

An action appears to be logical and promising—but in practice it has unintended counter-productive consequences to what was desired, or makes other things worse.

The acquisition archetypes adapt the systems thinking approach to describe the recurring patterns of counter-productive behavior in software-intensive acquisition programs. Each of the acquisition archetypes relates the story of a real-world acquisition program that experienced the dynamic, describes how that dynamic occurs on programs more generally, provides a causal loop diagram that can be used to analyze it, and recommends some of the ways the behavior can be mitigated and prevented.

In the following sections three different acquisition archetypes are discussed: Underbidding the Contract, Firefighting, and the Bow Wave Effect. Each one is presented with a summary description of the archetypes, accompanied by a causal loop diagram that depicts the dynamic behavior. A fourth section gives an example of how these archetypes can interact on a program.

Underbidding the Contract

In the "Underbidding the Contract" archetype shown in Figure 1, the use of the underbidding strategy to win contract awards is successful, and a reinforcing behavior sets in that increases the likelihood of future underbidding. While this approach may have

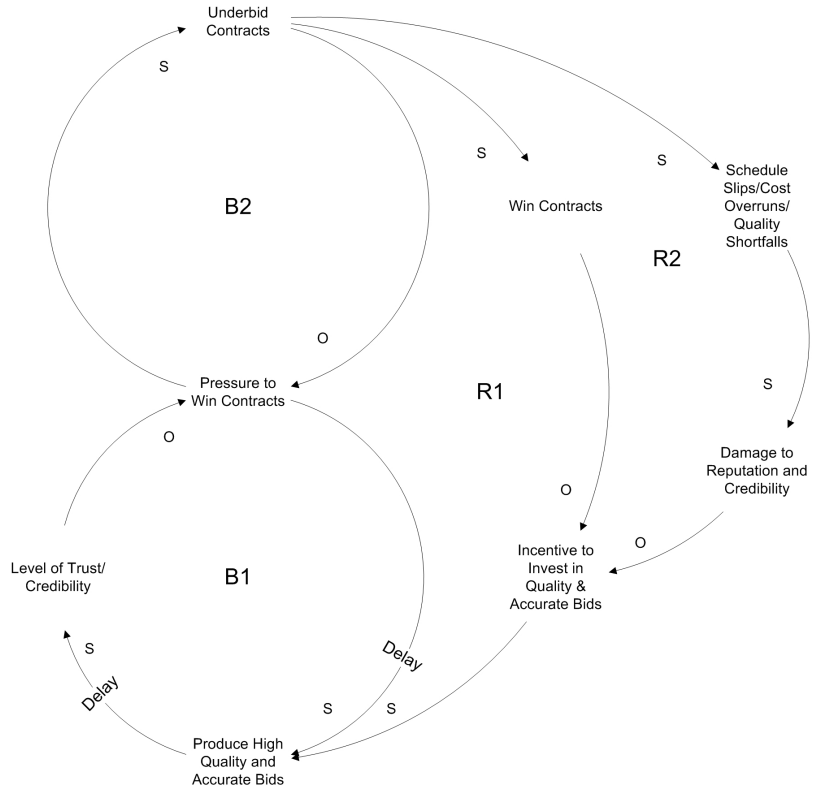

Figure 1: Overview of the "Underbidding the Contract" Dynamic[2, 3]

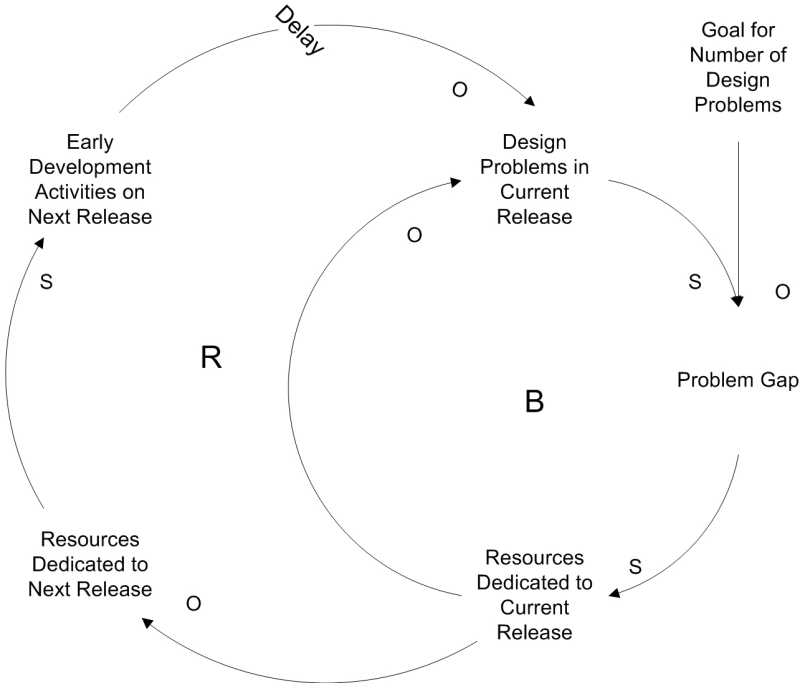

Figure 2: Overview of the "Firefighting" Dynamic4

some negative outcomes such as a damaged corporate reputation when the reality of the underbid becomes apparent, thus reducing any remaining intention to produce accurate bids—the advantage of having won the business may be enough to compensate for that. This seeming success is likely to then encourage other contractors to use the strategy themselves to stay competitive, because accurate bids may not be as successful at winning business.

Firefighting

In the "Firefighting" archetype that is shown in Figure 2, when a program has a target for the number of allowable defects in the delivered system, and finds itself exceeding that threshold, developers may be shifted from doing early design work on the next release of the system to fixing defects in the current release—which solves that problem. However, unless the total development staff is increased, the lack of designers working on the next release will unavoidably introduce problems into that release. When the next release becomes the current release, it will have even more defects, and the cycle will continue and worsen.

The Bow Wave Effect

The "Bow Wave Effect" archetype shown in Figure 3 shows a pattern of decisions in spiral development which are intended to improve visible progress by postponing riskier tasks in favor of more straightforward tasks that have a higher likelihood of being completed successfully in the near-term. While this approach does improve apparent progress, a backlog of complex tasks that have been deferred to a later spiral is building up like the bow wave in front of a large ship. These tasks will eventually have to be implemented at a time when more of the system has been built, there is less flexibility to accommodate changes, the program may be short on time and budget, and is less able to mitigate the risks those complex tasks may pose.

Interactions Among Archetypes

Many of the acquisition archetypes are related to one another, and may interact in predictable ways. In most actual programs, multiple interconnected archetypes are seen playing out simultaneously. The diagram in Figure 4 shows one possible set of these interactions.

Initial schedule pressure is created from underestimating effort (underbidding the contract) in order to win the contract. As the schedule pressure increases, a decision is made to delay some of the riskier tasks to be able to show better initial progress to management (the bow wave effect), but in actuality planting a time bomb that the program will trigger late in the development lifecycle when there is no time available to absorb the risk of those tasks. As the schedule starts to slip, certain quality shortcuts begin to occur (missed code reviews, etc.) as a way of reducing the workload and making up time. The increased defects resulting from the weakened quality processes inject new defects into the software—which add to the workload and divert developers from development to bug-fixing (firefighting). With diverted developers, productivity slows, further increasing schedule pressure, and continuing the cycle.

Solving Problems

The primary value of the acquisition archetypes is that they provide a model of the mechanism by which dynamic behaviors occur in systems. Without a model, or with an incorrect model, any proposed solutions to avoid or mitigate the behavior will not address the true root causes, and will be ineffective at best—and disastrous at worst. Lasting improvement will only come from changing the underlying system structure. The causal loop diagrams of the archetypes can be used to make explicit the points at which the dynamic can be influenced so as to improve

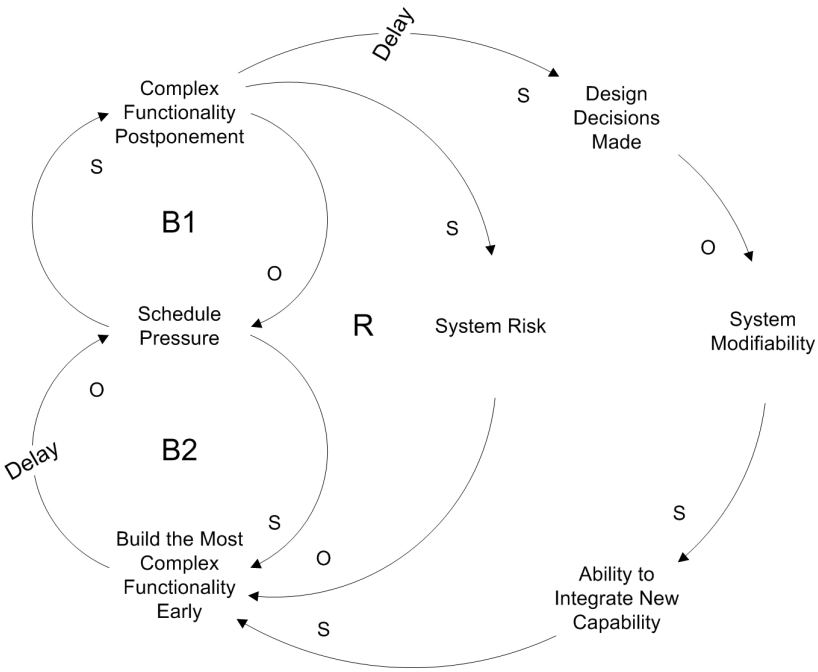

Figure 3: Overview of the "Bow Wave Effect" Dynamic5

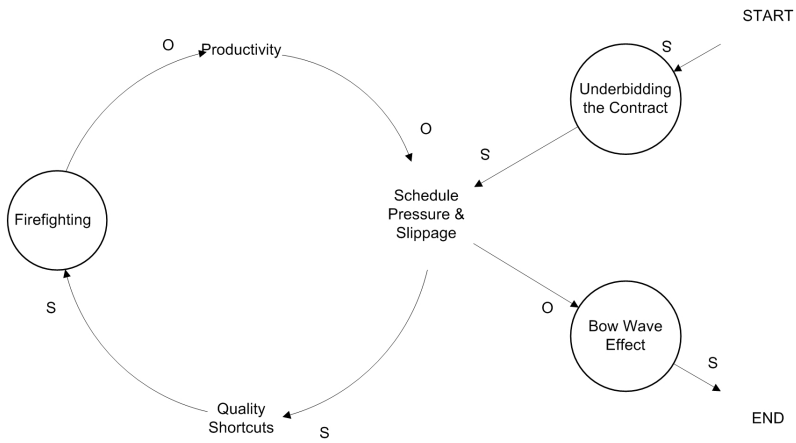

Figure 4: Diagram of Underbidding/Firefighting/Bow Wave Effect Archetypes

the typical outcome. With the aid of a causal loop diagram of the situation there are various techniques that can be used to mitigate adverse dynamics. Some of these described by the authors and Daniel Kim are outlined below:

- Reverse the direction of the archetype: It may be possible to turn negative (i.e., adverse) dynamics into positive ones by "running them backwards" and making them beneficial.
- Slow unwanted reinforcing loops: This approach follows the adage, "When you are in a hole, stop digging." While this approach will not eliminate the problem, it will help to minimize the damage, and buy time.
- Accelerate desirable reinforcing loops: The idea here is to make an already beneficial dynamic into one that has even more positive impact.
- Change the value around which a balancing loop stabilizes: In some cases what makes a balancing loop problematic is not

its behavior per se, but rather the specific value around which it stabilizes. In such cases we can change the equilibrium value to be something more acceptable.

- Shorten the duration of a time delay: Make it easier to manage the dynamic by bringing the cause and effect closer together in time, to make the linkage between them more evident.
- Find points where a small input change can produce a large effect: Because of a complex system dynamic known as the "Butterfly Effect6," small changes to the inputs of a complex, dynamic system can drive large changes in the outputs. Look for places in the diagram where small interventions can be leveraged by the feedback.
- Identify instances of social dilemmas and apply appropriate candidate solutions: Leverage prior solutions that have been identified.

Applying these techniques to the example archetypes described here provide some practical approaches to breaking out of, or preventing the "Underbidding the Contract" archetype, which include:

- Requiring full technical detail in the Request For Proposal, and thoroughly evaluating proposals
- Investing in, and trusting, a credible government cost estimate
- Establishing a new, realistic cost baseline and replan
- Restructuring the contract
- Looking for tip-offs that underbidding is occurring, such as staff productivity levels that are unrealistic
- Weighting the total technical value of the offer far above bid price in the proposal

Some possibilities for correcting and precluding the firefighting dynamic include:

- Realizing that diverting resources to fix defects only alleviates the symptoms—not the underlying problem—and committing to fix the real problem, with good estimates and more staff
- Revising the plan and/or schedule
- Avoiding investments in new approaches (i.e., improving staff productivity) if the organization is already resource-constrained.
- Doing resource planning with a view across the entire project, rather than locally

Potential ways of mitigating or avoiding the "Bow Wave Effect" archetype include:

- Stopping the use of expedient solutions, but doing so gradually, rather than all at once
- Identifying the root cause for choosing the expedient solution, and changing those incentives
- Considering only options that the organization can realistically handle

Beyond these approaches, the benefit of identifying a problem as an instance of a social dilemma, such as the case of the joint program described previously, is that there is a large set of mitigations and solutions that has been developed to address them.

There are three categories of solutions to social dilemmas:

- **Motivational:** Encourage people to want to change their behavior, because they are concerned about the possible impacts of their actions on others
- **Strategic:** Give people a reason to change their behavior that benefits themselves as well as the larger group
- **Structural:** The most difficult type to implement, the goal is to change the rules of the situation so that people must change their behavior—but this requires some level of authority to implement it, can engender resistance, and may require more expensive compliance enforcement

The motivational and strategic classes of solutions do not require changing the fundamental structure of the situation, and are thus simpler to implement, although potentially less effective than a structural solution.

Motivational solutions, while generally having a lower cost, work best when the participants have little self-interest, which is rarely the case in larger-scale software acquisition programs.

A strategic approach would be to make small changes to the incentive and reward structure of the program, such as improving communications, and making negative behaviors more apparent. While no single such change may significantly mitigate the problem, the aggregate effect of many small changes taken together could have a substantial positive impact. Strategic solutions, however, rely on reputations in longer-term relationships, which are problematic for shorter-tenure active duty servicemen.

The use of a central authority to manage the shared resource (i.e., "commons") at the heart of a social trap is a widespread structural approach, especially in government and military systems where such approaches are already frequently used. However, this approach has unintended side effects such as the incentive it provides to find creative "loopholes" in the mandate (such as a broad interpretation of the definition of "compliance" with the mandate).

There are many other solutions to addressing social dilemmas, such as building trust, exclusion mechanisms, rewarding group achievement (rather than just individuals), and assurance contracts. The choice of the best solution will depend on the specific circumstances surrounding the specific social dilemma.

Conclusions and Future Work

This paper has described a set of example acquisition archetypes that underlie the problems faced by the acquirers and developers of complex software-intensive systems, along with a set of recommended approaches for resolving them. The hope is that this set of acquisition archetypes will be used to help improve acquisition program performance, and that additional research work can be done to produce more acquisition archetypes in the future.

In the future, as the relationships and dynamic effects within programs grow more complex and interact, people will be less able to model the feedback mechanisms of the organizations in their heads to see what the larger emergent effects might be. Fortunately, there are other ways to analyze counter-productive patterns of behavior in programs. The SEI is

exploring the development of system dynamics models of software acquisition programs that can simulate the behaviors of such programs. One potential application for decision-making in acquisition programs is the development of interactive educational tools such as management flight simulators to help train acquisition program staff to understand these types of situations better, and thus be better equipped to manage them more effectively.

Another possible application of such a computer model is to answer a question frequently raised by acquisition program leaders: "How will a given change impact the program in terms of cost, schedule, scope, and quality?" It is not feasible to conduct experiments on larger-scale development efforts to answer these kinds of questions. However, a general-purpose, tailorable system dynamics model could help answer such hypothetical "what if?" scenario questions by providing a qualitative analysis of specific program contexts. Such a decision-support tool could improve the quality of key decisions made in acquisition programs—where even small, incremental improvements could provide better program outcomes and substantially improved value and cost savings for the Department of Defense (DoD).

While much more remains to be done to produce better acquisition outcomes, it is hoped that the approaches outlined here can be further developed and applied more broadly to achieve that goal.

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.

NOTES

1. The original story of the "Tragedy of the Commons" envisions a group of 19th century herders sharing an area of grazing land called a commons. If one herder decides to graze an extra animal, then that herder receives more benefit from the commons than the others, and at no additional cost to himself. However, if all of the herders follow suit, and add more animals according to the same reasoning, they eventually reach the point where the grass is eaten faster than it can grow, the cattle begin to starve, and ultimately all of the herders lose their livelihood.
2. Causal loop diagrams show how system variables (nodes) influence one another (arrows). The effects of the arrows are labeled "S" for "Same" when both variables change in the same direction, or "O" for "Opposite" when the variables change in opposite directions. Loops formed by the arrows are labeled either "B" for "Balancing" when they converge toward a value, or "R" for "Reinforcing" when they continually increase or decrease. The term "Delay" on an arrow indicates an actual time delay.
3. This diagram is based on the "Shifting the Burden" systems archetype described in (Senge, 1990).
4. This diagram is the "Firefighting" dynamic described in (Repenning, Goncalves, & Black, 2001).
5. This diagram is based on the "Shifting the Burden" systems archetype described in (Senge, 1990).
6. The "Butterfly Effect" refers to the sensitivity of the outputs of a deterministic, nonlinear system to very small changes in the inputs. It was named by the mathematician and meteorologist Edward Lorenz, and refers to the theoretical possibility of a hurricane forming as the result of a butterfly flapping its wings.

ABOUT THE AUTHORS

William E. Novak is a Senior Member of the Technical Staff at the Carnegie Mellon University Software Engineering Institute, with over thirty years of experience with government software systems acquisition and real-time embedded software. Mr. Novak held positions with GE Corporate Research and Development, GE Aerospace, Texas Instruments, and Tartan Laboratories. Mr. Novak received his M.S. in Computer Engineering from Rensselaer Polytechnic Institute, and B.S. in Computer Science from the University of Illinois at Urbana-Champaign.

E-mail: wen@sei.cmu.edu
Phone: 412-268-7700

Andrew P. Moore is a Senior Member of the Technical Staff at Carnegie Mellon University's Software Engineering Institute with more than 25 years of experience in mission-critical systems modeling and analysis. He has worked for the Naval Research Laboratory and has published widely, including a book on insider cybersecurity threats. Andy received a MA in Computer Science (Duke University), a BA in Mathematics (College of Wooster), and a Graduate Certificate in System Dynamics (Worcester Polytechnic Institute).

E-mail: apm@sei.cmu.edu
Phone: 412-268-7700

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213
(412) 268-7700

REFERENCES

1. Cross, John G. and Melvin J. Guyer. Social Traps. Ann Arbor: University of Michigan Press, 1980.
2. Firefighting. Dir. William E. Novak. Software Engineering Institute. 2012. Animated Short. <http://www.sei.cmu.edu/acquisition/research/archetypes.cfm>.
3. Forrester, Jay W. Principles of Systems. Pegasus Communications, 1971.
4. Hardin, Garrett. "Tragedy of the Commons." Science 162 (1968): 1243-1248.
5. Kadish, Ronald. "Defense Acquisition Performance Assessment Report - Assessment Panel of the Defense Acquisition Performance Assesment Project." 2006.
6. Kim, Daniel H. System Archetypes: Diagnosing Systemic Issues and Designing High-Leverage Interventions. Vols. I, II, & III. Pegasus Communications, Inc., 1993. 3 vols.
7. Kollock, Peter. "Social Dilemmas: The Anatomy of Cooperation." Annual Review of Sociology 24 (1998): 183-214.
8. Madachy, Raymond J. Software Process Dynamics. Wiley-IEEE Press, 2008.
9. Meadows, Donella. Thinking in Systems: A Primer. White River Junction, VT: Chelsea Green Publishing, 2008.
10. Moore, Andrew P. and William E. Novak. "Modeling the Evolution of a Science Project in Software-Reliant System Acquisition Programs." Conference of the Systems Dynamics Society. Boston, MA, 2013. <http://www.systemdynamics.org/conferences/2013/proceed/papers/P1029.pdf>.
11. Moore, Andrew P. and William E. Novak. "The Joint Program Dilemma; Analyzing the Pervasive Role that Social Dilemmas Play in Undermining Acquisition Success." Proceedings of the 10th Annual Naval Postgraduate School Acquisition Research Symposium. Monterey, CA, 2013. <o http://www.acquisitionresearch.net/files/FY2013/NPS-AM-13-C10P01R07-036.pdf>.
12. Novak, William E. and Harry L. Levinson. "The Effects of Incentives in Acquisition Competition on Program Outcomes." Proceedings of the Defense Acquisition University Acquisition Research Symposium. Ft. Belvoir, VA, 2012. <http://www.sei.cmu.edu/library/abstracts/reports/12tr001.cfm>.
13. Novak, William E. and Linda Levine. Success in Acquisition: Using Archetypes to Beat the Odds. Technical Report. Software Engineering Institute. Pittsburgh, PA, 2010. <http://www.sei.cmu.edu/library/abstracts/reports/10tr016.cfm>.
14. Novak, William E., Andrew P. Moore and Christopher Alberts. The Evolution of a Science Project: A Preliminary System Dynamics Model of a Recurring Software-Reliant Acquisition Behavior. SEI Technical Report . Carnegie Mellon University. Pittsburgh: Software Engineering Institute, 2012.
15. Repenning, Nelson P., Paulo Goncalves and Laura J. Black. "Past the Tipping Point: The Persistence of Firefighting in Product Development." Califonial Management Review (2001).
16. Senge, Peter M. The Fifth Discipline. Doubleday/Currency, 1990.

The Bow Wave Effect. Dir. William E. Novak. Software Engineering Institute. 2013. Animated Short. <http://www.sei.cmu.edu/acquisition/research/archetypes.cfm>.