# Understanding the Drivers Behind Software Acquisition Program Performance

*Enabling Mission Success through Improved Software Decision-Making*

Andrew P. Moore
William E. Novak

April 10, 2013

# The Tar Pit

"Large-system programming has over the past decade been… a tar pit, and many great and powerful beasts have thrashed violently in it. Most have emerged with running systems—few have met goals, schedules, and budgets. Large and small, massive or wiry, team after team has become entangled in the tar. No one thing seems to cause the difficulty—any particular paw can be pulled away. But the accumulation of simultaneous and interacting factors brings slower and slower motion. Everyone seems to have been surprised by the stickiness of the problem, and it is hard to discern the nature of it. But we must try to understand it if we are to solve it."

*—Frederick Brooks, The Mythical Man-Month*

# Introduction

“At its very core, this acquisition business is not about contracts, testing, acquisition strategies, plans, technology, finance, oversight, or any of the other things one can learn about or make rules about. It's about people.”

*—Terry Little, Missile Defense Agency*

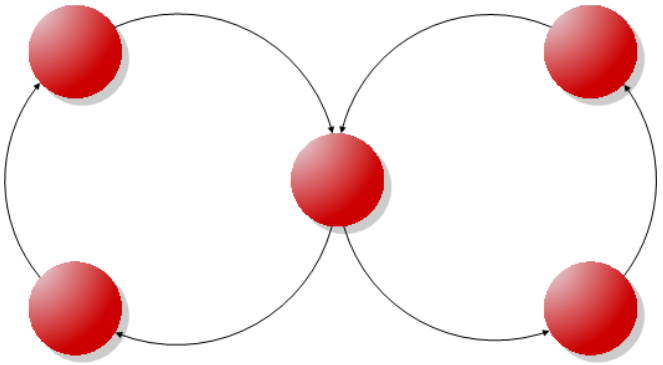
# Objectives

Introduce the *Acquisition Archetypes* and acquisition dilemmas

Show how programs can start to recognize, avoid, and resolve common counter-productive behaviors in software acquisition and development

Influence how acquisition practitioners and leaders make decisions

Present a different way of approaching/resolving acquisition problems



# Agenda

**Introduction**

Misaligned Incentives and Structural Dynamics

Social Dilemmas

**Systems Thinking**

Systems Archetypes

**Acquisition Archetypes**

Acquisition Dynamics

Learning Games for Acquisition

**Breaking the Pattern**

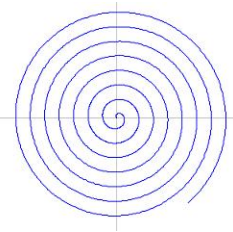
Solving Social Dilemmas

**Conclusions and Summary**

# Did You Ever Wonder…

Why do acquisition programs believe it's possible to make up schedule by cutting corners on development?

Why do programs violate spiral development by doing the riskiest development *last*?



Why do investments in failing acquisition programs continue long past the point that makes economic sense?

Why is it that "Win/Win" partnerships degenerate for no apparent reason?

Why do some of a program's most critical risks or issues never make it to the attention of the program manager?

Why, with advanced estimation models, do large programs underestimate costs by up to 70%?

# Why is Software-Intensive Acquisition So Hard?

**Complex interactions** between PMO, contractors, sponsors, and users
- The full chain of actions and their longer-term consequences is not clear

**Limited visibility** into real progress and status
- Hard to apply corrective actions when status is uncertain

**Significant delays** exist between applying changes and seeing results
- Difficult to control systems with long delays between cause and effect
- *Examples*: Reorganizing a department, Steering an aircraft carrier

**Unpredictable** and unmanageable progress and results
- Complexity of interdependencies has unintended consequences

**Uncontrolled escalation** of situations despite best management efforts
- Misaligned incentives can drive potentially conflicting behaviors

**Linear partitioning** is the standard approach to address large systems
- When systems have feedback between components that are partitioned, it makes it difficult to see and address these interactions

**Exponential growth of interactions** as size grows linearly

# …Because It's a Complex, Dynamic System!

**Organizational**: Key issues in software acquisition are often management and organization-related — *not* technical—and people mean feedback

- "No matter what the problem is, it's always a people problem."
  —*Gerald Weinberg*

**Complex Interactions**: Interactions between acquisition stakeholders are *non-linear* because of the presence of feedback

- *What* you *do depends on what* I *do, which depends on what* you *do…*

**Non-linear Behavior**: Feedback defies traditional mathematical analysis

**Sensitivity to Initial Conditions**: Results may vary greatly due to very small differences in starting point(s)

**Partitioning**: Partitioning isn't possible when there are complex interactions between components

# Can Systems Trap Us into Behaviors?

Inside a complex, dynamic system, people's actions can be at the mercy of that system's dynamics. Such patterns occur in real estate cycles:

As price drops…

→ demand increases (*get a good deal*)

→ …*and after a delay*… (*takes time to buy*)

→ supply decreases (*not many houses left*)

→ price *increases* (*supply and demand*)

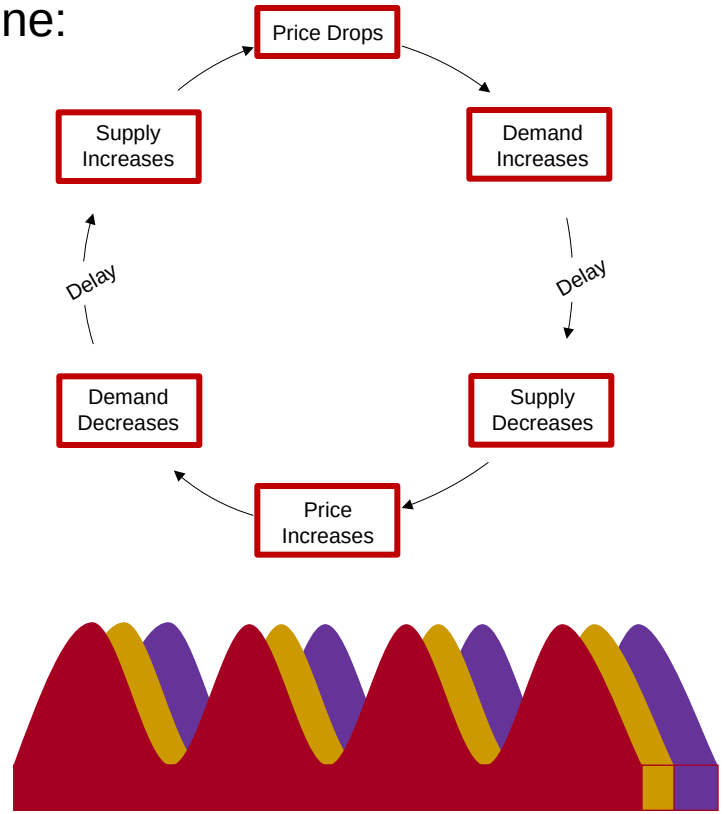
→ demand *decreases* (*too expensive now)*

→ …*and after a delay*… (*more people must sell*)

→ supply *increases* (*plenty of houses*)

→ and price drops… (*supply and demand*)

Since this is a *loop*, let's draw it as one:



# Misaligned Incentives and Structural Dynamics

"Incentives are misaligned—PMs and contractors are not necessarily rewarded for decisions that lead to lower life cycle costs or provide a better balance between cost and performance"

*—Defense Acquisition Performance Assessment, GEN Ronald Kadish (Ret.)*

# Misaligned Incentives

Structural reasons like feedback and delays aren't the only causes for acquisition failure—*incentives* play a key role as well.

Misaligned incentives occur when:

- *Individual* goals conflict with *group* goals
- *Short-term* goals conflict with *longer-term* goals

The result is that:

- Some *group* goals only succeed at the expense of *individual* goals
- Some *longer-term* goals can only succeed at the expense of *short-term* goals

Some acquisition programs are *prevented* from succeeding for structural and incentive reasons—*not* poor work or lack of effort.

**Take-away**

Misaligned incentives can force people to make impossible choices.

# Misaligned Incentives in Acquisition

**Risk**: Low incentive to identify program risks if it can adversely affect personal standing

**Defects**: Incentives to find defects can result in the intentional insertion of defects

**Schedule**: Incentives to improve performance by meeting a set date can mean quality processes are sacrificed to meet that date

**Technology**: Incentives to use risky, immature technology to achieve better system capability, and give good experience to the contractor

**Contracts**: Incentives to drag out development on CP & T&M contracts to increase profits

**Staffing**: Incentives to slow efforts/stretch schedule if there's no next project to move on to

**Cancellation**: Low incentive to cancel ailing programs if it's not in interests of program staff

**Scope**: Low incentive for users to ask for only minimal system capability if it's free to them

**Take-away**

Misaligned incentives occur every day in every area of acquisition programs

# Social Dilemmas

"Social traps are baited [with]… positive rewards which… direct behavior along lines that seem right every step of the way, but nevertheless end up at the wrong place."

*—John Cross and Melvin Guyer, Social Traps*

"Morality boils down to self-interest. People cooperate where their outcomes are correlated."

*—Robert Wright*

## Social Dilemmas

# The Prisoner's Dilemma

Two suspects are arrested by the police, who don't have enough evidence to convict either—so they separately offer each the same deal. If one "*rats*" and the other stays silent, the rat goes free and his accomplice gets 10 years. If both stay silent, then both get 6 months on a minor charge. If each one "*rats*," then *each* gets 5 years.

Each prisoner must choose to "rat" or keep quiet. Each one is told that the other won't hear about him "ratting" before the end of the investigation. What should they do?

# Prisoner's Dilemma Payoff

| **Prisoner's Dilemma (*Player 1, Player 2*)** | | **Player 1** | |
|---|---|---|---|
| | | Cooperate (Silence) | Defect ("Rat") |
| **Player 2** | Cooperate (Silence) | (0.5, 0.5) *1 year total* | (0, 10) *10 years total* |
| | Defect ("Rat") | (10, 0) *10 years total* | (5, 5) *10 years total* |

# Social Dilemmas

What if we all could be better off, but no one has an incentive to change?

Dilemmas are all about cooperation—and there are two basic types:

## The Tragedy of the Commons

- Someone wants a *benefit* that will *cost* everyone else
- Some are tempted by that benefit, but if *all* do, everyone is worse off.

## Producing a Public Good

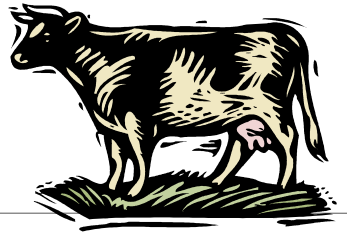
- Someone faces a near-term *cost* that would *benefit* everyone else
- Some try to avoid the cost, but if *all* do, everyone is worse off.

**Key Idea**

The “Tragedy of the Commons” is a multi-player version of the “Prisoners’ Dilemma”

## Social Dilemmas

# The Tragedy of the Commons



The “Tragedy of the Commons” refers to a pasture area shared by farmers.

It works as long as you don’t graze too many cattle, so the grass can grow back.

One farmer might graze more cattle to make more money—but if everyone does the same, the grass is destroyed, the cattle starve, and everyone loses.

Free access and unlimited demand for a finite resource dooms the resource through exploitation.

The concept behind the Tragedy of the Commons is real, and can be seen in:

- **Overfishing**: Everyone wants to catch more fish—but if everyone does, there will be no more fish
- **Congestion**: Everyone using a car because it’s more convenient creates traffic jams—so it’s less convenient for everyone
- **Polluting**: It’s cheaper to pollute—but everyone *else* pays the price

**Key Idea**

“Individually optimal decisions lead to collectively inferior solutions.”

# Systems Thinking

"We human beings do not see the larger system processes of which we are a part."

*—Barry Oshry, Seeing Systems*

# What is Systems Thinking?

Systems Thinking is a method for analyzing complex systems

Developed by Jay W. Forrester at MIT while modeling electrical feedback

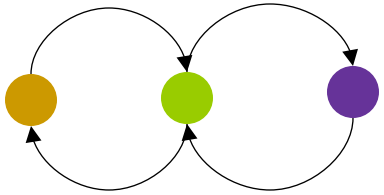
- Also exists in economic, political, business, and organizational behaviors

Uses feedback loops to analyze common system structures that either spin out of control, or regulate themselves

Helps identify a system's underlying structure, and what *actions* will produce which *results* (and *when*)

Systems Thinking teaches us that:

- *System behavior is greater than the sum of component behaviors*
- "Quick fix" solutions usually have side-effects that can make things worse
- True improvement comes from changing the underlying system structure



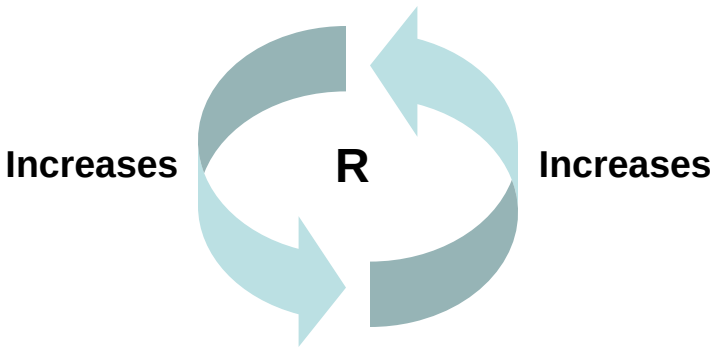
# Causal Loop Diagrams (CLDs)

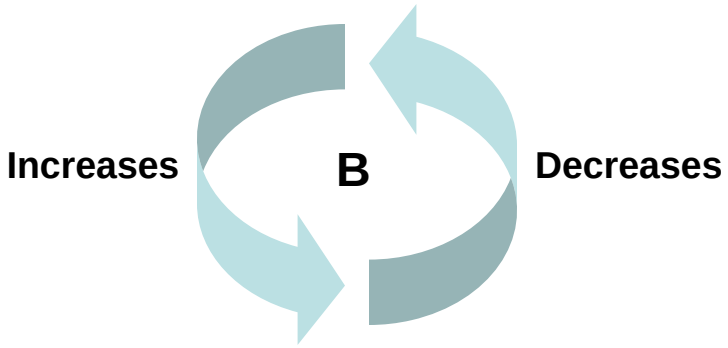
Depict qualitative "*influencing*" relationships (increasing or decreasing) and time delays between key variables that describe the system

Show relationship direction by labeling them **S**ame (**+**) or **O**pposite (**-**) to indicate how one variable behaves based on the previous variable

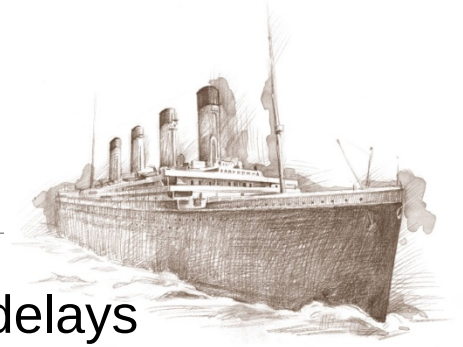
Consist primarily of two types of feedback loops:

- **R**einforcing – Changes to variables *reinforce*, moving in one direction
- **B**alancing – Changes to variables *alternate*, achieving equilibrium



# Time Delays



Much unpredictability of systems is due to time delays

Time delays *obscure* the connections in cause-and-effect relationships

- Side-by-side causes and effects would be “smoking gun” evidence

People are poor at controlling systems with big time delays between the cause and the effect

- *Example*: Over-steering a large ship that is slow to respond, so it weaves back and forth
- *Example*: A temperature control on a low-BTU air conditioner that’s slow to cool, so the temperature bounces between too hot and too cold
- *Example*: Can’t determine which surface, handshake, sneeze, or cough caused you to get sick—so it’s hard to avoid catching something

This happens in companies and acquisition programs as well:

- *Example*: If the expected benefits of a reorganization or improvement aren’t visible in a short time, it’s assumed that it didn’t help

# Emergent Behavior

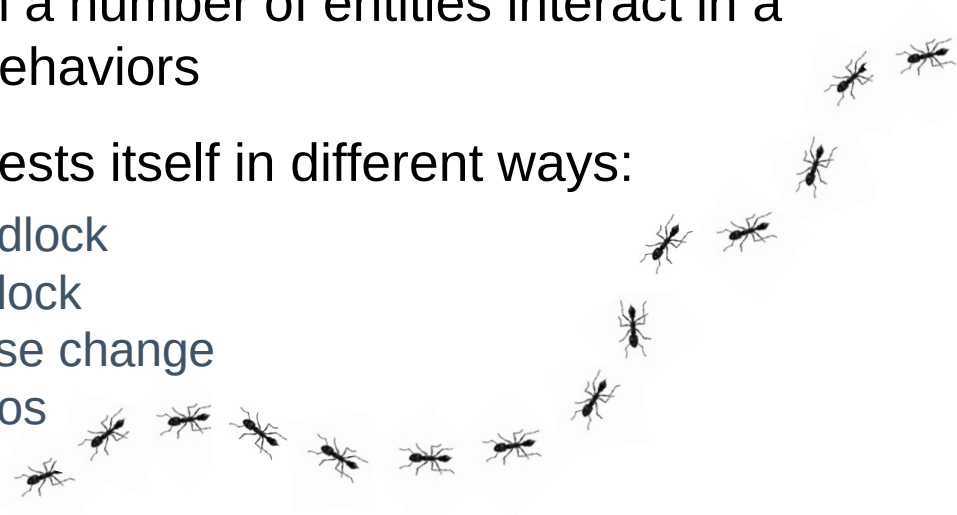
"The arising of novel and coherent structures, patterns and properties during the process of self-organization in complex systems."

- *Prof. Jeffrey Goldstein, Adelphi University: Emergence*

An *emergent behavior* appears when a number of entities interact in a system, collectively producing new behaviors

Emergent behavior in systems manifests itself in different ways:

- Oscillation
- Escalation and decline
- Synchronization
- Thrashing
- Deadlock
- Livelock
- Phase change
- Chaos



Examples of emergent behavior

- The ebb and flow of traffic, the flocking of birds, evolving patterns of cities/suburbs, synchronized clapping, market sell-offs, ant foraging, etc.

**Key Idea**

The systems archetypes are *all* examples of emergent behavior.

# The Butterfly Effect

Complex systems can be highly sensitive to small changes in initial conditions—producing results and behavior that *appear* to be random

- In short, two runs of the same system using similar starting conditions can produce vastly different outcomes

The name *The Butterfly Effect* came from meteorologist Edward Lorenz, who suggested that "a butterfly flapping its wings in Brazil could ultimately produce a tornado in Texas."

Examples of the Butterfly Effect:

- A change in an African animal virus is believed to have spread to human beings and created the AIDs epidemic
- While the movements of a ball in a pinball machine are precisely governed by physics, small variations in friction and mechanics make its path virtually unpredictable
- The losses in the U.S. subprime mortgage sector depressed housing markets globally, causing turmoil in international financial markets.

# Systems Archetypes

"Whether or not what you do has the effect you want, it will have three at least that you never expected, and one of those will usually be unpleasant."

*—Robert Jordan*

# What are the Systems Archetypes[1]?

We're good at recognizing problems when they arise, and dealing with them—but we don't always recognize familiar problems if they look *different*.

If we don't recognize the similarity of problems, we treat each one as if it's new.

Over 10 systems archetypes have been identified which represent common patterns of behavior that recur across many disciplines.

Each one tells a story that sounds all too familiar, but they share a common pattern:

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

System archetypes show the structures that lay underneath some of our most challenging problems—structures that offer ways of resolving them.

[1] *from Peter Senge, The Fifth Discipline: The Art and Practice of the Learning Organization, Doubleday, 1990.*

# Systems Archetypes -1

## Fixes that Fail

- A quick fix for a problem has immediate positive results, but its unforeseen long-term consequences worsen the problem.
- *Example*: Using credit cards to pay off debt
- *Example*: Stopping a course of antibiotics when you're feeling better

## Balancing Loop with Delay

- A system's state is moving toward the desired state through repeated action, but the delay raises doubts about its effectiveness.
- *Example*: Real estate cycles
- *Example*: Adjusting the temperature of a shower

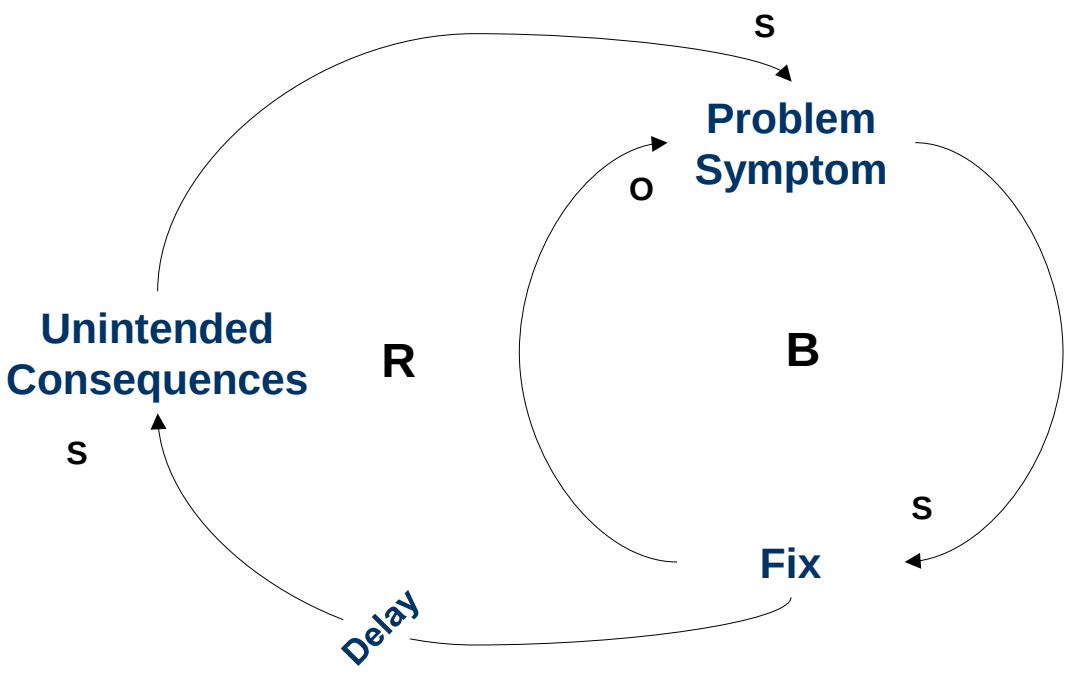
## Limits to Growth

- Initially rapid growth slows because of an unseen inherent capacity limit in the system that worsens with growth.
- *Example*: Town stops growing when traffic becomes intolerable
- *Example*: Overpopulation and limited food production

*from Peter Senge, The Fifth Discipline: The Art and Practice of the Learning Organization, Doubleday, 1990.*

## Systems Archetype

# "Fixes That Fail (Backfire)"




A quick *Fix* for a *Problem Symptom* has immediate positive results, but also has long-term *Unintended Consequences* that, after a *delay*, worsen the original *Problem Symptom* as the *Fix* is used more often.

*from Daniel H. Kim, "System Archetypes: Vols. I, II, and III. Pegasus Communications, 1993.*

# Systems Archetypes -2

## Shifting the Burden ("Addiction")

- An expedient solution temporarily solves a problem, but its repeated use makes it harder to use the fundamental solution.
- *Example*: Dependence on coffee, rather than sleep, to stay awake
- *Example*: Welfare acting as a substitute for good employment

## Accidental Adversaries

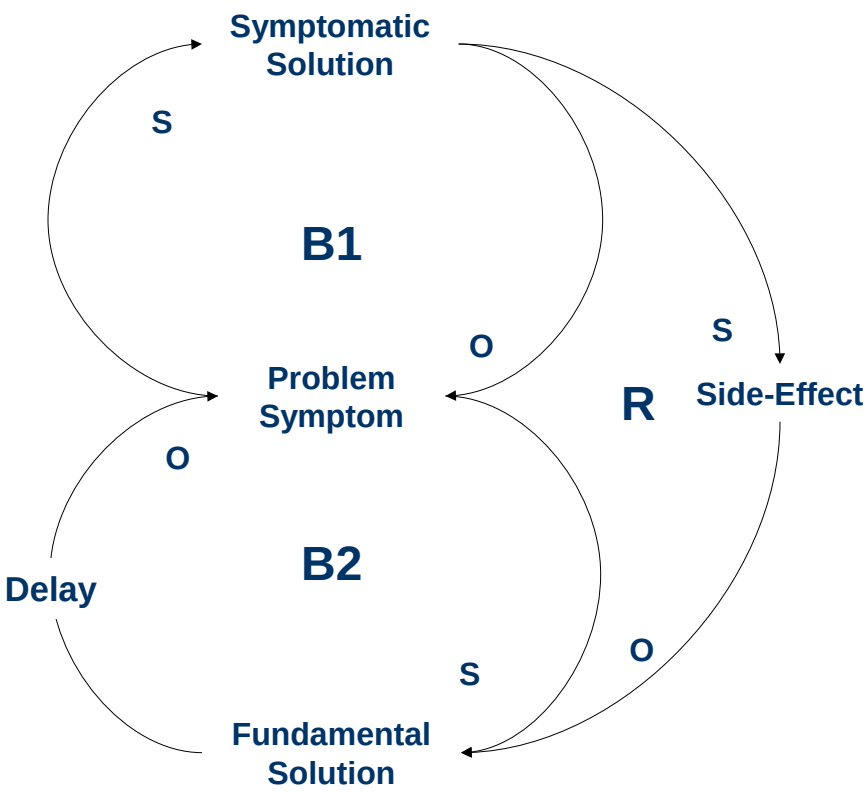
- Two cooperating parties destroy their relationship through escalating retaliations for perceived injuries.
- *Example*: Initially happy marriage ultimately leads to divorce
- *Example*: Conflict between short-term sales & long-term research goals
- *Example*: Failed mergers—British Airways/USAir, Daimler-Chrysler

## Escalation

- Two parties compete for superiority, with each escalating its actions to get ahead.
- *Example*: The nuclear arms race between the U.S. and the U.S.S.R.
- *Example*: Price wars between adjacent competing stores

## Systems Archetype

# "Shifting the Burden"




A *Symptomatic Solution* temporarily solves a *Problem Symptom*, which later recurs. Its repeated use over the longer term has *Side-Effects* that make it less and less feasible to use the more effective *Fundamental Solution*—trapping the organization into using only the *Symptomatic Solution*. Impatience with the delay makes the organization choose the *Symptomatic Solution* in the first place.

*from Daniel H. Kim, "System Archetypes: Vols. I, II, and III. Pegasus Communications, 1993.*

# Systems Archetypes -3

## Drifting Goals

- A gradual decline in performance or quality goals goes unnoticed, threatening the long-term future of the system.
- *Example*: Cutting the volume of soda in a can & selling it for the same price
- *Example*: Gradually replacing quality ingredients with artificial substitutes

## Growth and Underinvestment

- Investments in a growing area aren't made, so growth stalls, which then rationalizes further underinvestment.
- *Example*: People's Express airline collapse due to poor customer service
- *Example*: Learning the violin on your own, with disheartening progress

*from Daniel H. Kim, "System Archetypes: Vols. I, II, and III. Pegasus Communications, 1993.*

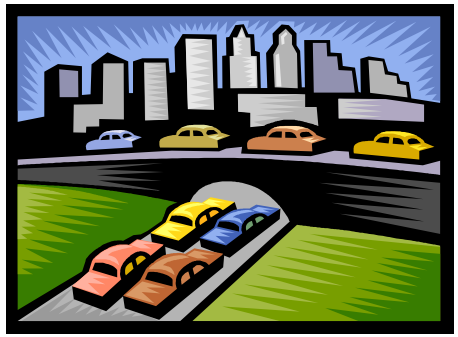
# Systems Archetypes -4

## Success to the Successful

- When two parties compete for a limited resource, the initially more successful party receives more resources, increasing its success.
- *Example*: BETAMAX vs. VHS
- *Example*: QWERTY keyboard layout
- *Example*: PC/Windows vs. Macintosh

## Tragedy of the Commons

- A shared resource is depleted as each party abuses it for individual gain, ultimately hurting all who share it.
- *Example*: Disappearance of sardines in Monterey Bay from overfishing
- *Example*: Congestion on urban highways



# Uses of System Archetypes

Identify failure patterns as they develop
- Realize there *is* a problem—do you see an archetype or incentives?

Single out root causes
- Diagnose the fundamental root causes of problems—not just symptoms

Engage in “big picture” thinking
- See the larger system, instead of just the piece you’re in

Promote shared understanding of problems
- Share a model of the problem with others who can help to solve it

Find interventions to break out of ongoing pattern
- Fix the pattern using leverage points from the structure

Avoid future counter-productive behaviors
- Prevent the most common traps simply by knowing about them

# Acquisition Archetypes

We're "…focused on the little, tiny swells and waves on the surface of the ocean. But in fact, most of the big things affecting the ocean are these currents underneath. They're what's moving the water."

*—John Sides, GWU*

# What are Acquisition Archetypes?

Acquisition archetypes are modeled on the systems archetypes

Acquisition archetypes are patterns of behavior seen time and again on actual programs that are counter-productive and undermine progress

Acquisition archetypes depict the underlying structures of the behaviors that occur throughout acquisition organizations

- Each causal loop diagram tells a familiar, recurring story
- Each describes the structure that causes the dynamic

*Acquisition Archetypes* are used to:

- Identify failure patterns as they develop (*recognition*)
- Single out root causes (*diagnosis*)
- Engage in “big picture” thinking (*avoid oversimplification*)
- Promote shared understanding of problems (*build consensus*)
- Find interventions to break out of ongoing dynamics (*recovery*)
- Avoid future counter-productive behaviors (*prevention*)

# Acquisition Archetypes -1

"Happy Path" Testing

- Schedule pressure drives "making up" lost time, which can cause shortcuts in quality (like testing, peer reviews, using coding standards…)

Firefighting

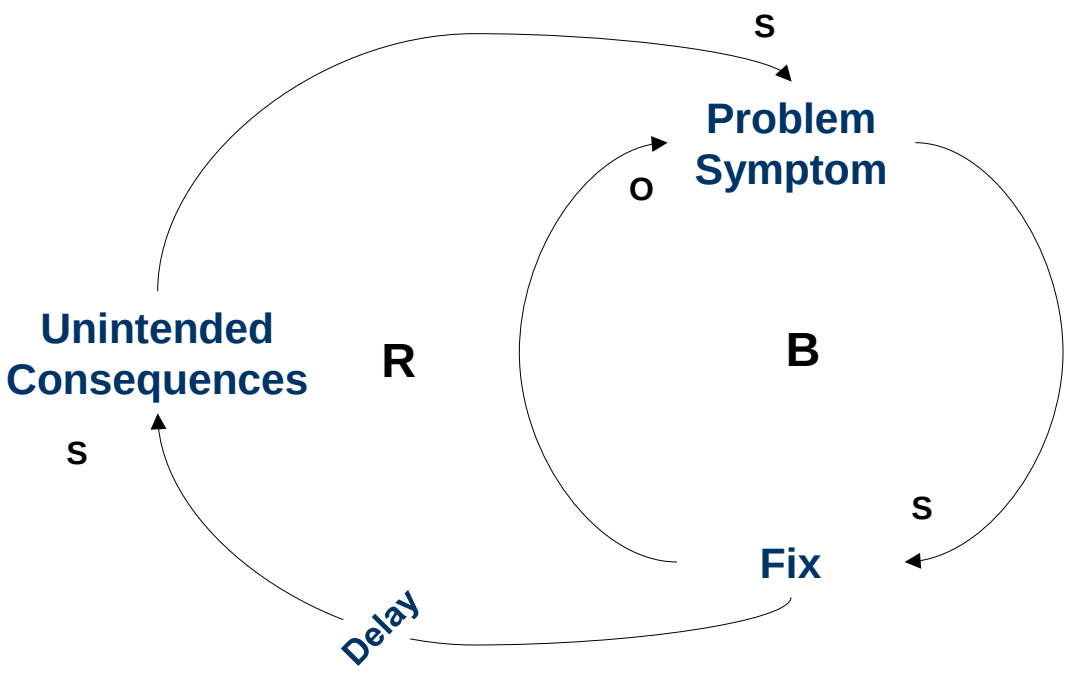
- Rework to fix defects in the current release diverts resources from the early design of the next release—injecting even *more* defects into it

Brooks' Law

- Adding new people to a late software project to speed development sounds attractive—but in reality causes *additional* delays

## Systems Archetypes

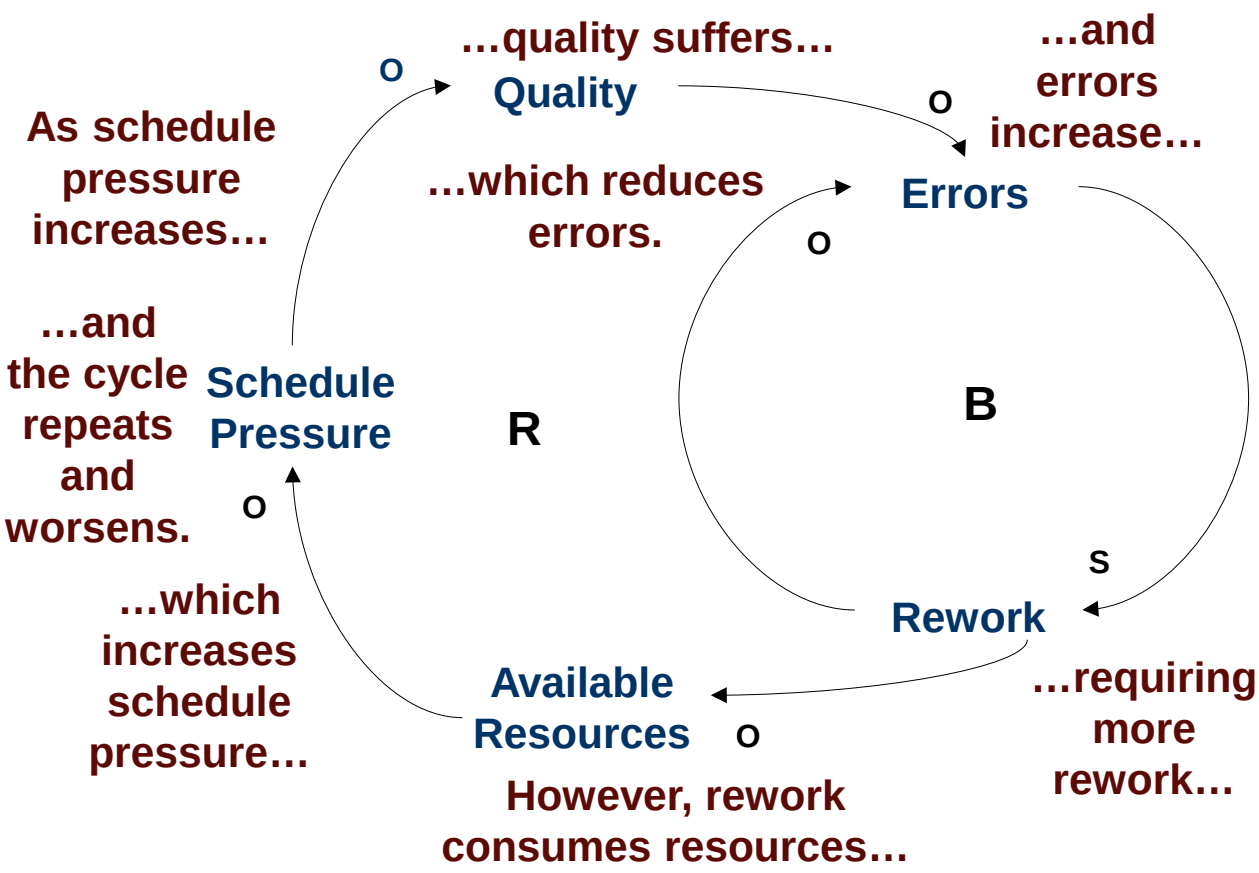
# "Fixes That Fail"




A quick *Fix* for a *Problem Symptom* has immediate positive results, but also has long-term *Unintended Consequences* that, after a *delay*, worsen the original *Problem Symptom* as the *Fix* is used more often.

*based on the "Fixes that Fail" systems archetype*

# "Happy Path Testing" (*i.e.*, *Sacrificing Quality*)

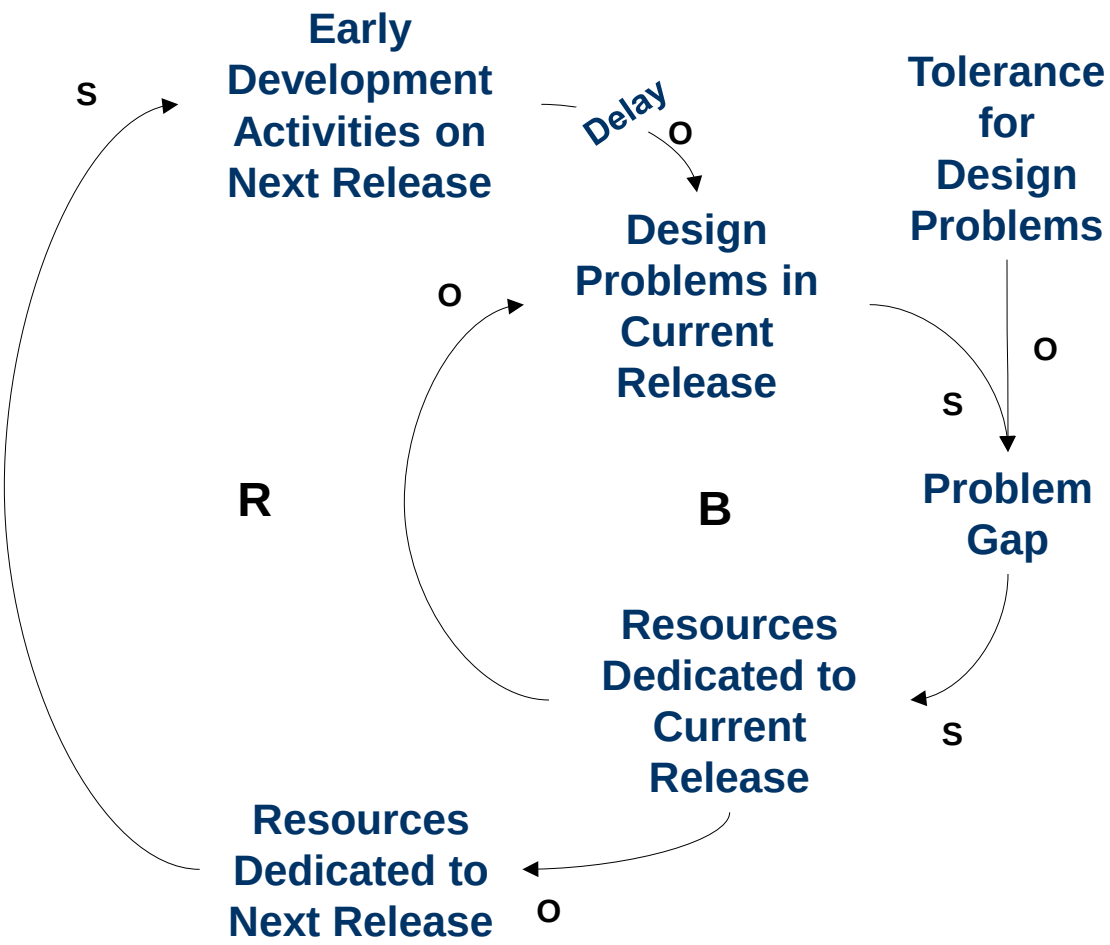



As schedule pressure increases, processes are shortcut, quality suffers, and errors increase—requiring more re-work. However, re-work consumes resources, which increases schedule pressure, and the cycle repeats and worsens.

*based on the "Fixes that Fail" systems archetype*

## Acquisition Archetypes

# "Firefighting"




If design problems in the current release are higher than the tolerance for them, then more resources must be dedicated to fix them. This reduces problems, but now fewer resources can work on the *next* release. This undermines its early development activities which, after a delay, increases the number of design problems in the next release.

*based on the "Fixes that Fail" systems archetype*

*from "Past the Tipping Point: The Persistence of Firefighting in Product Development," Repenning, Goncalves, & Black, 2001.*

# Acquisition Archetypes -2

The “Bow Wave” Effect

- Riskier tasks planned for an early development spiral are delayed in favor of simpler tasks—increasing risk by leaving less time, less budget, and less flexibility to address issues

Longer Begets Bigger

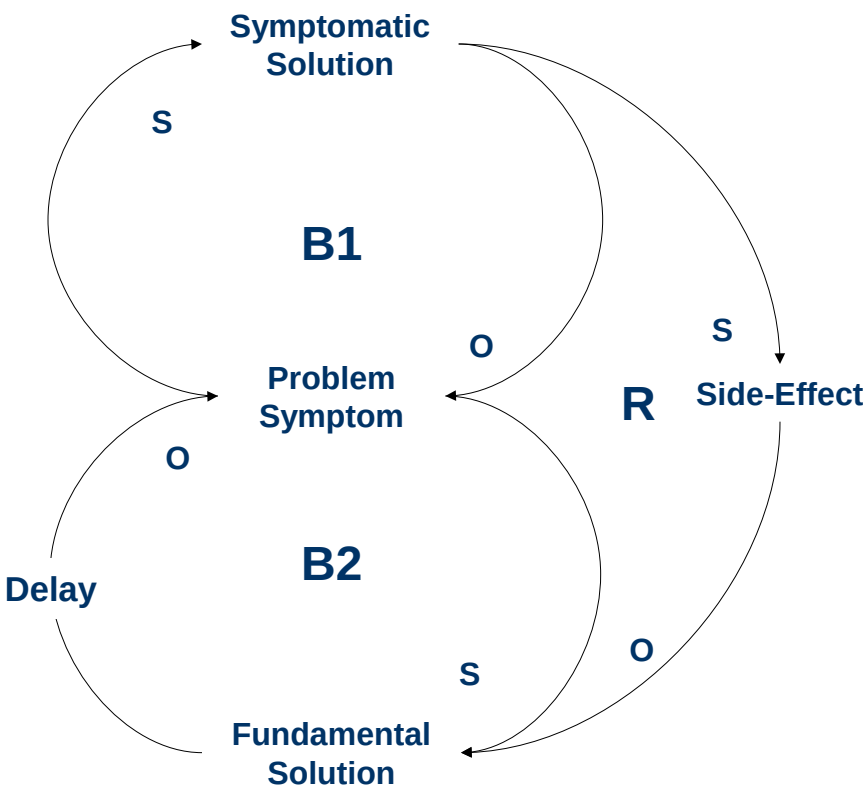
- Large program development causes lengthy schedules—during which technology and operational environment changes cause scope changes, resulting in even longer schedules and higher cost

Robbing Peter to Pay Paul

- By giving overspent programs extra funding taken from those that are underspent, overspenders succeed, underspenders fail, and overspending is perpetuated

## Systems Archetypes

# "Shifting the Burden"

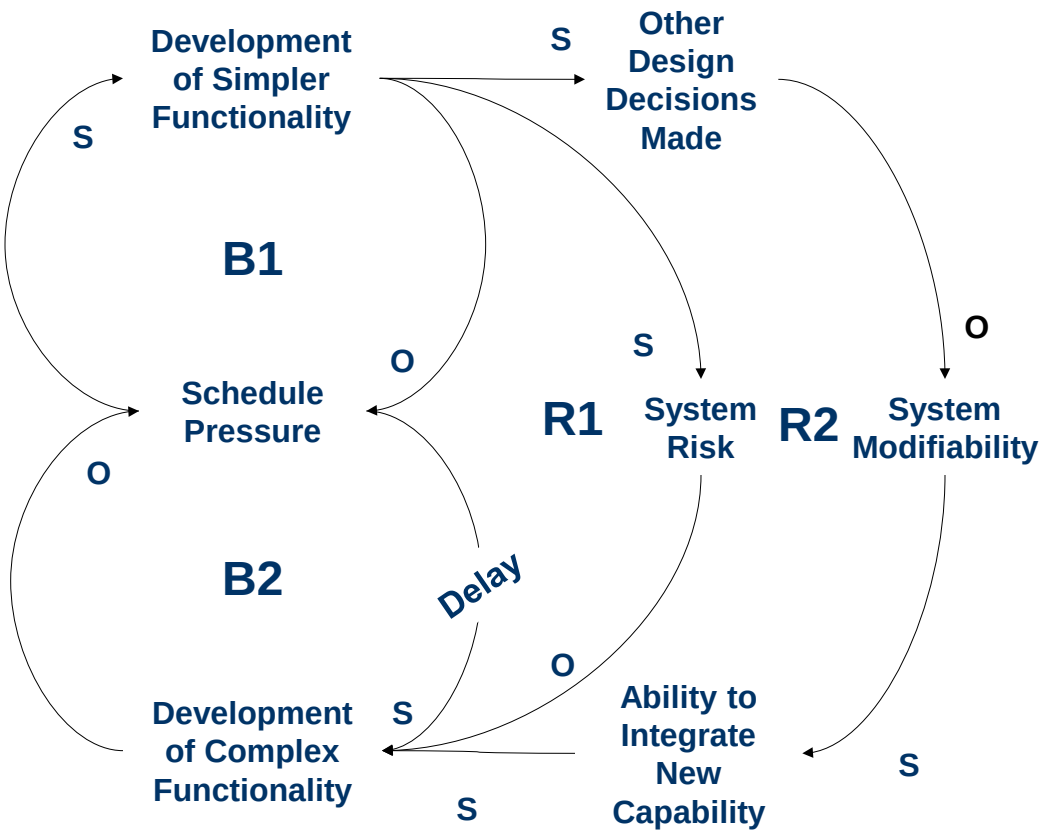



A *Symptomatic Solution* temporarily solves a *Problem Symptom*, which later recurs. Its repeated use over the longer term has *Side-Effects* that make it less and less feasible to use the more effective *Fundamental Solution*—trapping the organization into using only the *Symptomatic Solution*. Impatience with the delay makes the organization choose the *Symptomatic Solution* in the first place.

*from Daniel H. Kim, "System Archetypes: Vols. I, II, and III. Pegasus Communications, 1993.*

## Acquisition Archetypes

# "Bow Wave Effect"




Risky tasks planned for an early spiral to reduce risk are postponed to a later spiral, making near-term performance look better. This increases risk in subsequent spirals by delaying required risky development for which there is now less available schedule to address potential issues, and less flexibility in the system to accommodate changes needed to integrate the new capability.

*based on the "Shifting the Burden" systems archetype*

# Acquisition Archetypes -3

Everything for Everybody

- Building common infrastructure must reconcile competing requirements into one system—but this drives up cost, schedule, risk, and complexity, driving user programs away

Underbidding the Contract

- Underbidding contracts often results in winning them—and when problems occur more time and money are given, which encourages other contractors to do the same

Staff Burnout and Turnover

- Increasing pressure and long hours eventually lead to burnout and turnover—which reduce productivity and further increase schedule pressure

# Acquisition Archetypes -4

PMO vs. Contractor Hostility

- The seemingly “win-win” relationship between PMO and contractor degenerates when one party inadvertently harms the other—who then retaliates

Feeding the Sacred Cow

- Management ignores warnings of program failure due to uncertainty, and continues on—often long after it is no longer economically defensible

Shooting the Messenger

- Managers who report bad news to executives are not rewarded, but are often punished—making other managers even more reluctant to report issues

# Did You Ever Wonder…

Why do acquisition programs believe it's possible to make up schedule by cutting corners on development? ***Firefighting***

Why do programs violate spiral development by doing the riskiest development *last*? ***The Bow Wave Effect***

Why do investments in failing acquisition programs continue long past the point that makes economic sense? ***Feeding the Sacred Cow***

Why is it that "Win/Win" partnerships degenerate for no apparent reason? ***PMO vs. Contractor Hostility***

Why do some of a program's most critical risks or issues never make it the attention of the program manager? ***Shooting the Messenger***

Why, with advanced estimation models, do large programs underestimate costs by up to 70%? ***Underbidding the Contract***

# “Firefighting” Animation

# “The Bow Wave Effect” Animation

# Acquisition Dynamics

"... We are pawns in a game whose forces we largely fail to comprehend."

*—Dr. Daniel Ariely, Duke University*

## Acquisition Dynamics

# Joint Programs

1. A JPO PM has six stakeholder programs planning to use their joint infrastructure software…

2. …but each program demands at least one major feature be added to the software just for them.

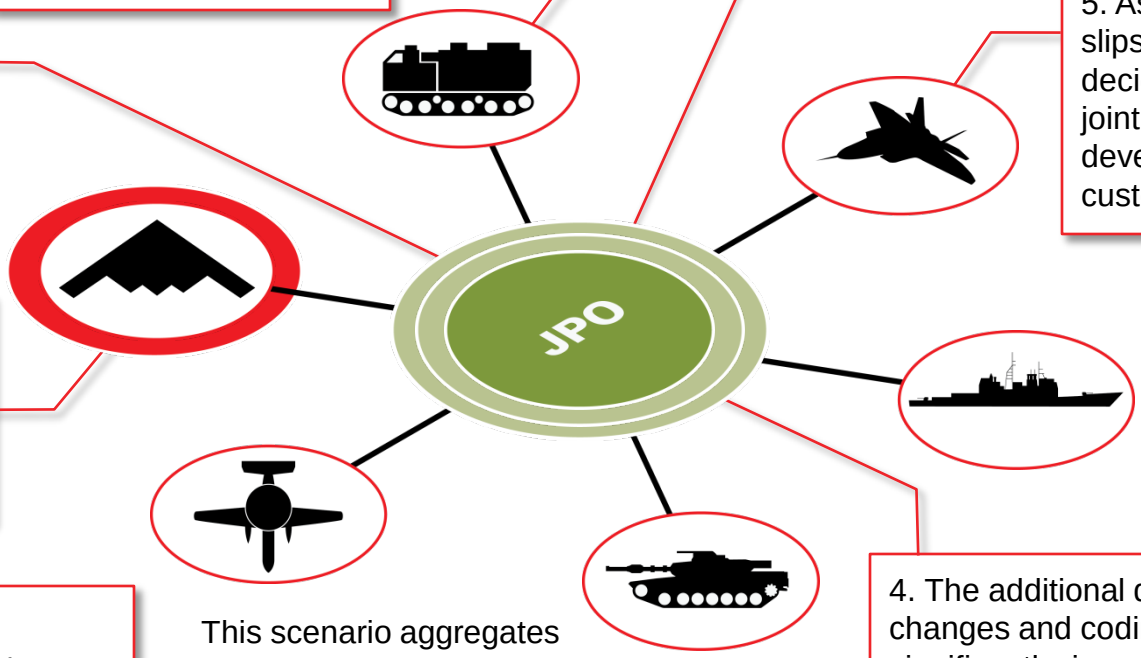
3. The JPO agrees to the additional requirements, for fear of losing stakeholders (who could build custom software).

4. The additional design changes and coding significantly increase total cost, schedule, complexity, and risk.

5. As the schedule slips, one program decides to leave the joint program and develop its own custom software.

6. With one stakeholder gone, the amortized costs for the other programs increase further—and another program leaves.

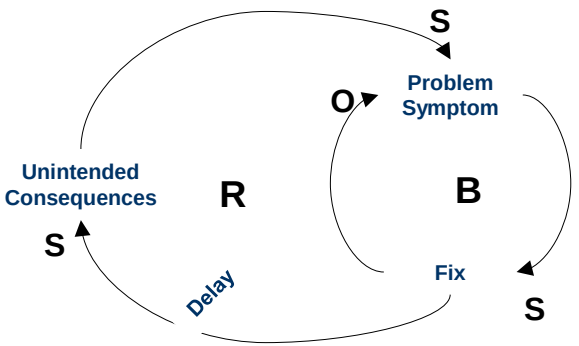
7. As cost escalates and schedules lengthen, participation in the joint program unravels and collapses.



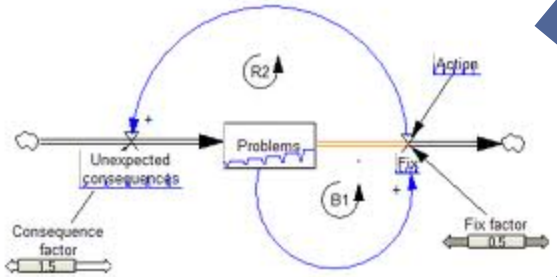

This scenario aggregates three SEI software-reliant system acquisition ITAs conducted in 2006-2009.

# Research Approach -1

General Qualitative Model




Acquisition Problem Model




Deep Understanding of Dynamic Acquisition Behavior

Model-Based Simulation of Potential Solutions

Basis for Acquisition Instructional Simulations

Acquisition Qualitative Model

Independent Technical Assessment (ITA) Data

Detailed examinations of challenged programs with interviews, document reviews, and code analysis

Firefighting: If design problems are found in the current release, more resources must be used to fix them. This reduces problems, but now less work is done on the *next* release. This undermines its early development work, and increases design problems in the next release.

# Research Approach -2

Build models of Acquisition Archetypes to create executable simulations of significant adverse acquisition program behaviors

- Turn existing software acquisition domain expertise into a more usable form

Use acquisition models to analyze known adverse software acquisition dynamics, and test proposed solutions

- Apply new and known solutions to solving recurring dilemmas in acquisition

Use experiential learning from hands-on simulations to give DoD acquisition staff a deeper understanding of acquisition dynamics to help make better decisions

- Understand common side-effects of decisions that lead to poor performance
- Let acquisition staff gain experience through education—not costly mistakes

Build foundation acquisition model to test value of future solution approaches

- Qualitatively validate new approaches before applying them to programs

# The Evolution of a Science Project

"What they did at first was a proof of concept, a quick and dirty prototype, and when they tried to scale it up, there were indications that it might not be possible…"

*—Acquisition Program Lead*

## Acquisition Dynamics

# The Evolution of a "Science Project"

9. Warfighters wait years for a new system to be built from scratch.

8. New versions of the system can't be deployed with needed capability, robustness, and performance.

7. New program office unwilling to discard prototype code due to field deployment pressures.

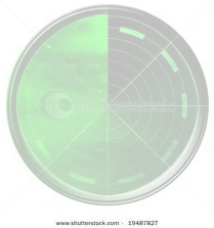
6. Project infrastructure, processes, & staff not able to scale up to production development.

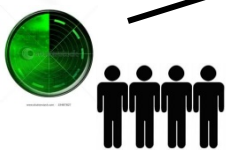
5. As system grows, poor architecture, documentation, & code quality cause poor reliability, performance, & usability.

3. Warfighters and field commanders demand more capability, broader deployment, faster response.

2. Prototype is deployed on small scale, and is well received.

1. Project begins as small informal effort to build prototype & prove concept.





4. Project staff is diverted to field support, so development progress slows.

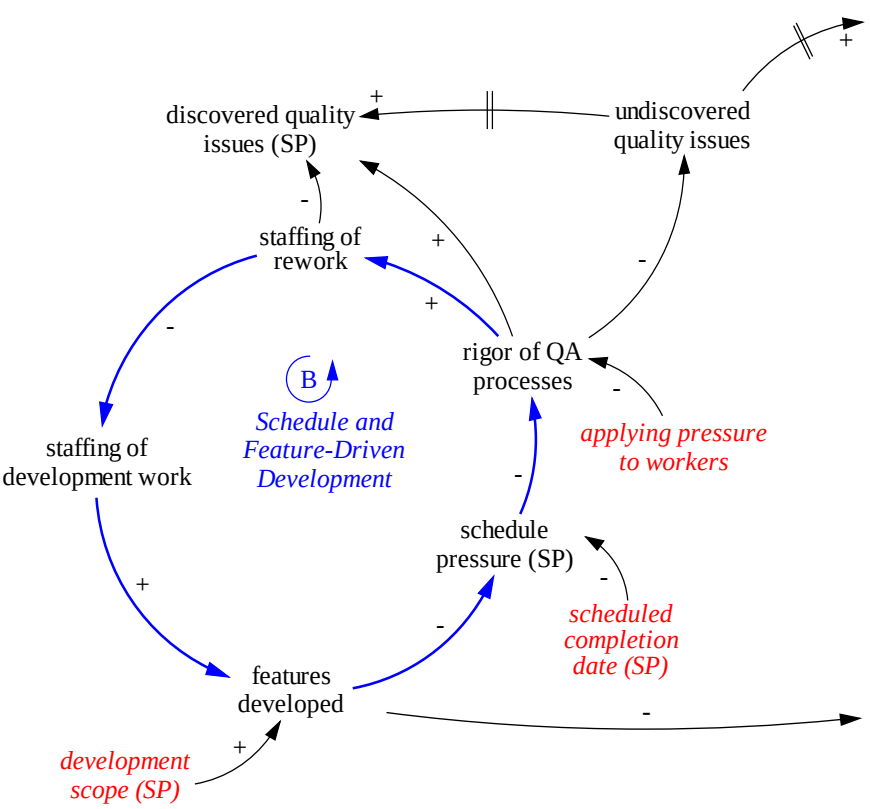
This scenario aggregates five SEI software-reliant system acquisition ITAs conducted in 2006-2009.

# The Evolution of a "Science Project"

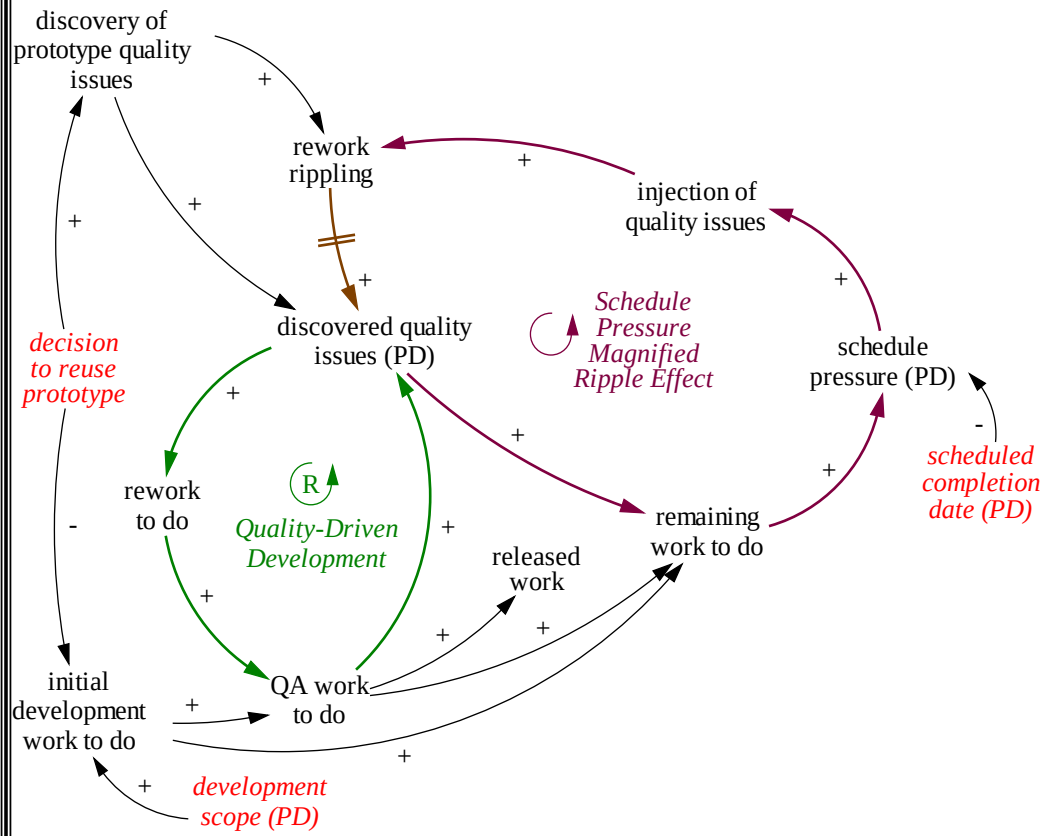
- This behavior has been recognized in many different programs
  - Acquisition executives have seen this dynamic play out in their portfolios
- Model was developed using VenSim system dynamics modeling package
- *Technical Report*: "The Evolution of a Science Project: A Preliminary System Dynamics Model of a Recurring Software-Reliant Acquisition Behavior"

# The Evolution of a "Science Project"

## Science Project (SP) Sector



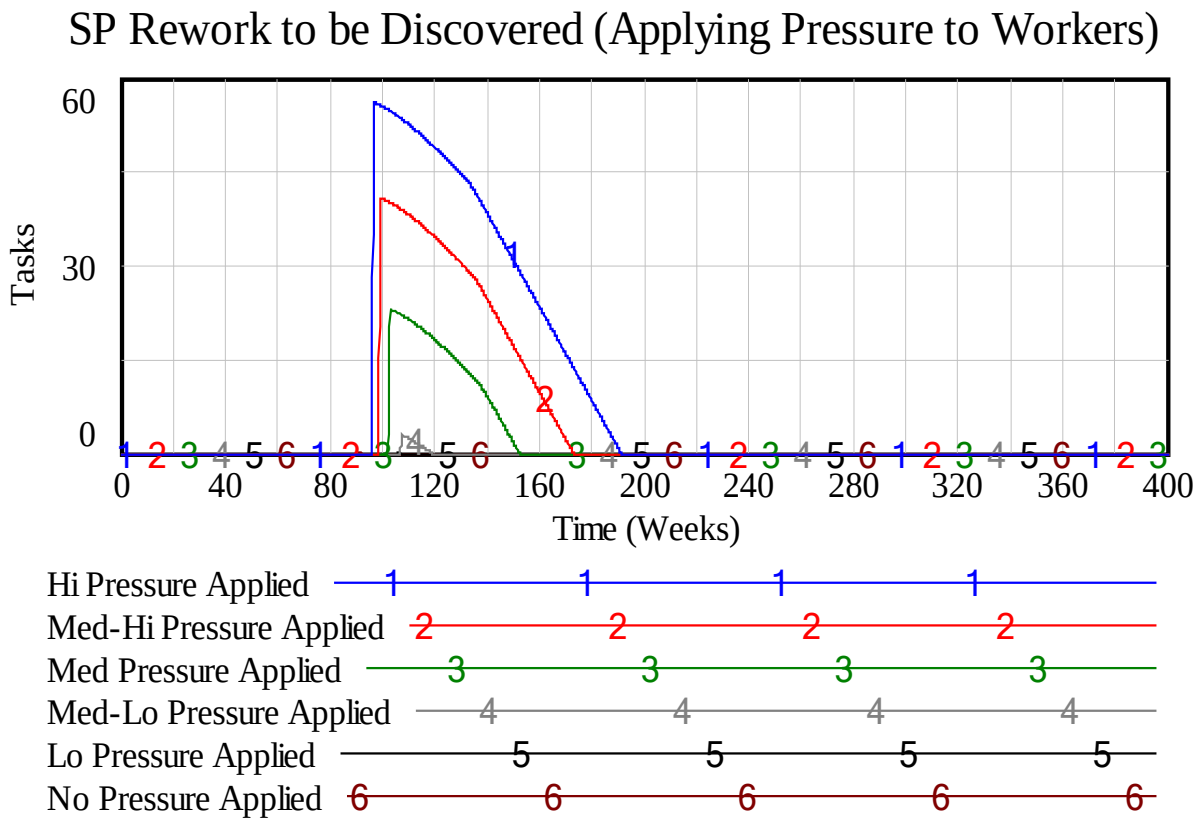

## Production Development (PD) Sector



# Key Preliminary Findings

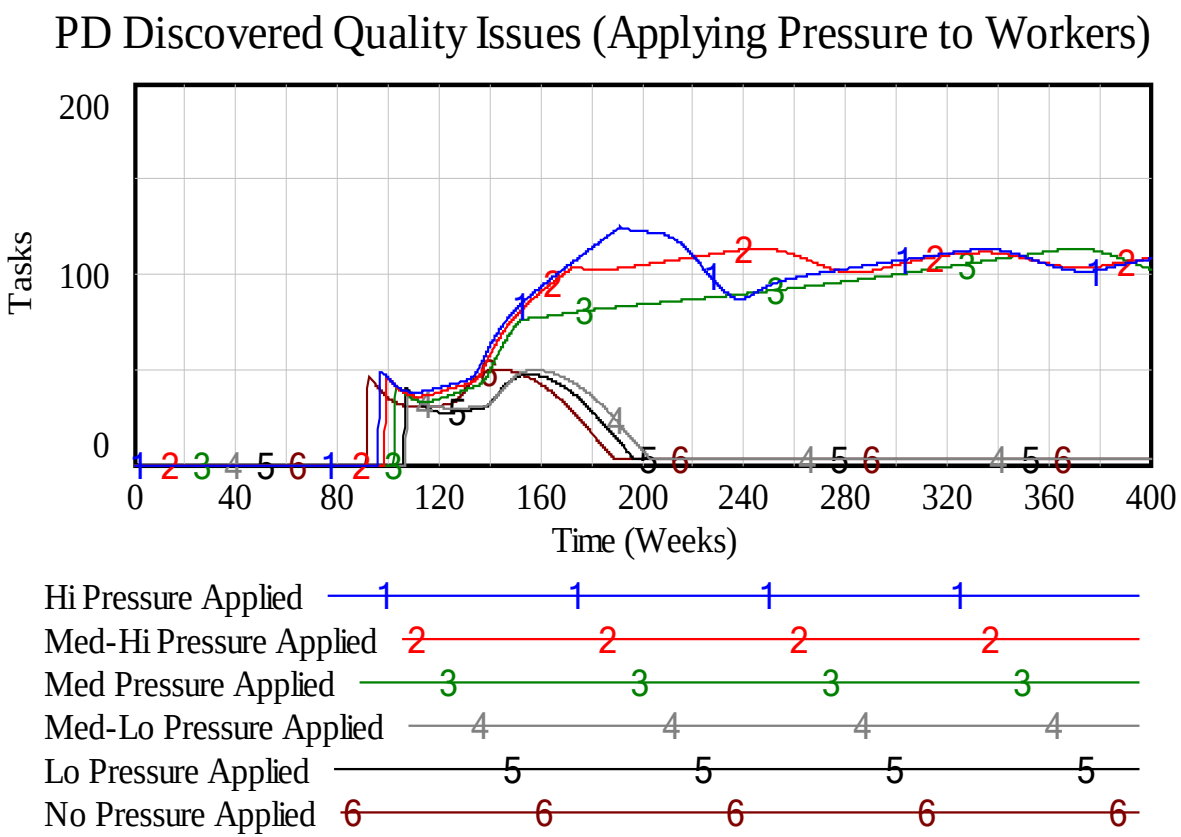
# Assumption

Applying pressure to workers developing SP results in undiscovered rework



# Key Preliminary Findings -1

High pressure, or moderate pressure for long periods, can lead to a “tipping point”



# The Tipping Point in Evolution of a Science Project

- Accumulating rework creates a dangerous feedback dynamic
- "Firefighting" due to rework is a key underlying element
- Key drivers in reaching the "tipping point" are:
  - a) pressure on developers
  - b) the degree of "ripple effect"
  - c) the emphasis on schedule and features vs. quality
  - d) the timing of the transition from science project to production development

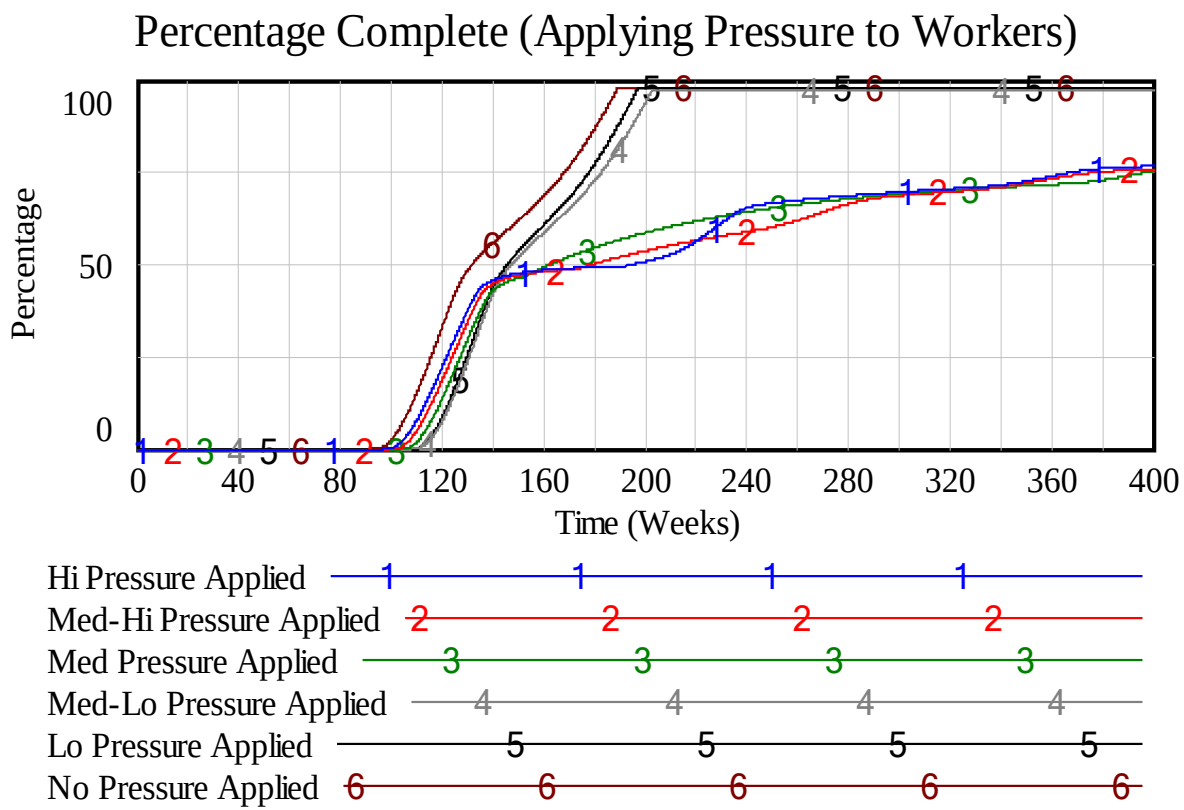
# Key Preliminary Findings -2

Placing modest pressure on developers for limited periods shortens schedule

- VenSim optimization shows that placing pressure at a low level is optimal with respect to reducing project duration
- By allowing periods of pressure, followed by periods of relaxation, the program might:
  - Limit worker burnout
  - Perform better regarding schedule

# Key Preliminary Findings -3

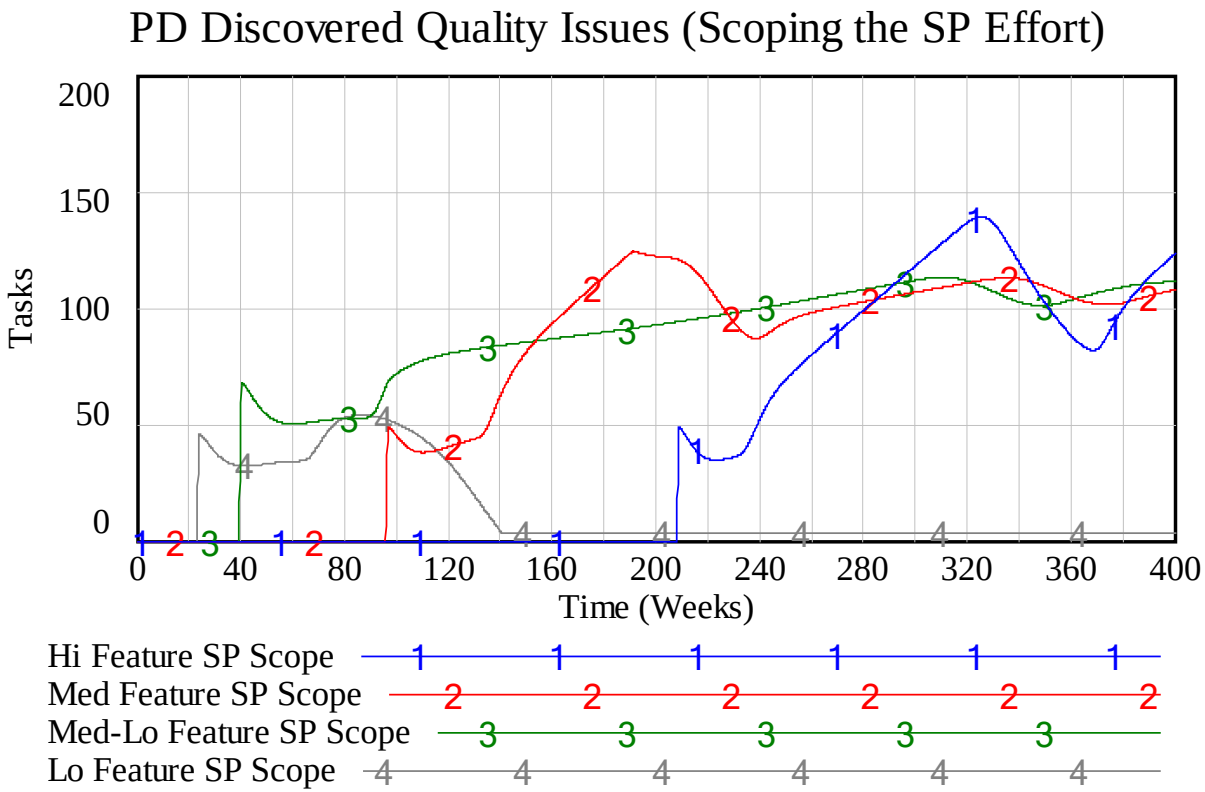
The tipping point contributes to the “90% Done” Syndrome



# Key Preliminary Findings -4

The transition from science project to production effort should be made *early*

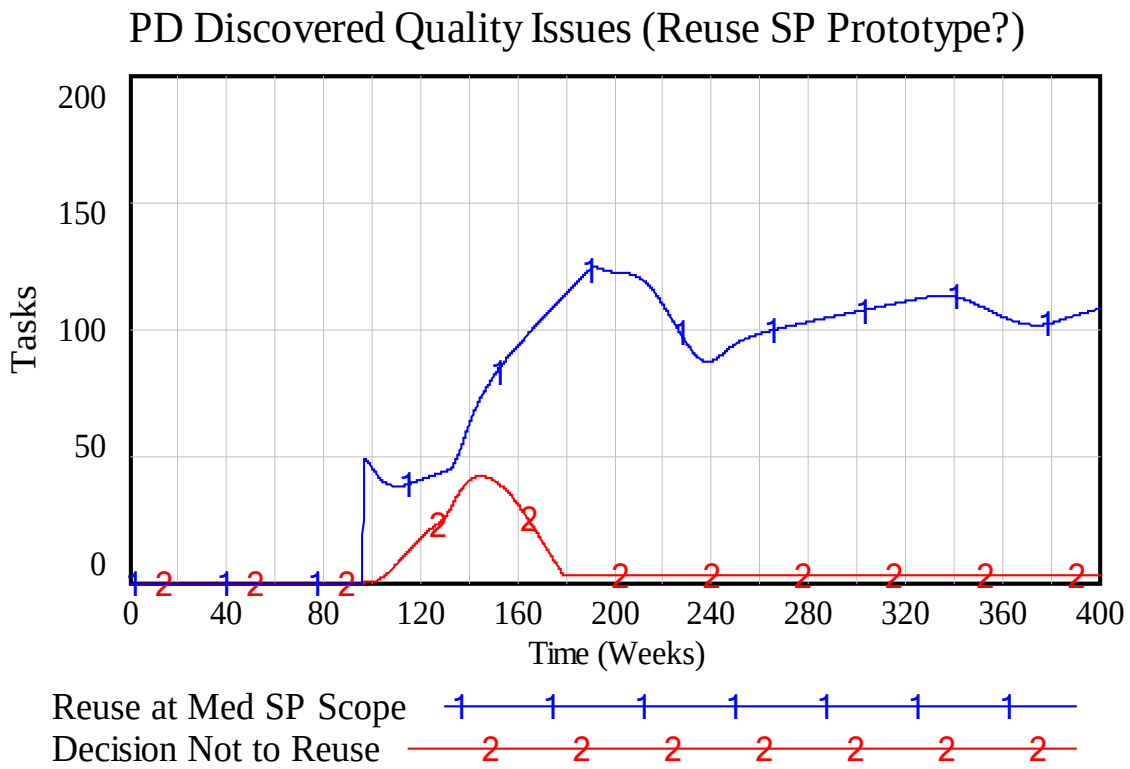
- A late transition increases the amount of undiscovered rework that is transferred



# Key Preliminary Findings -5

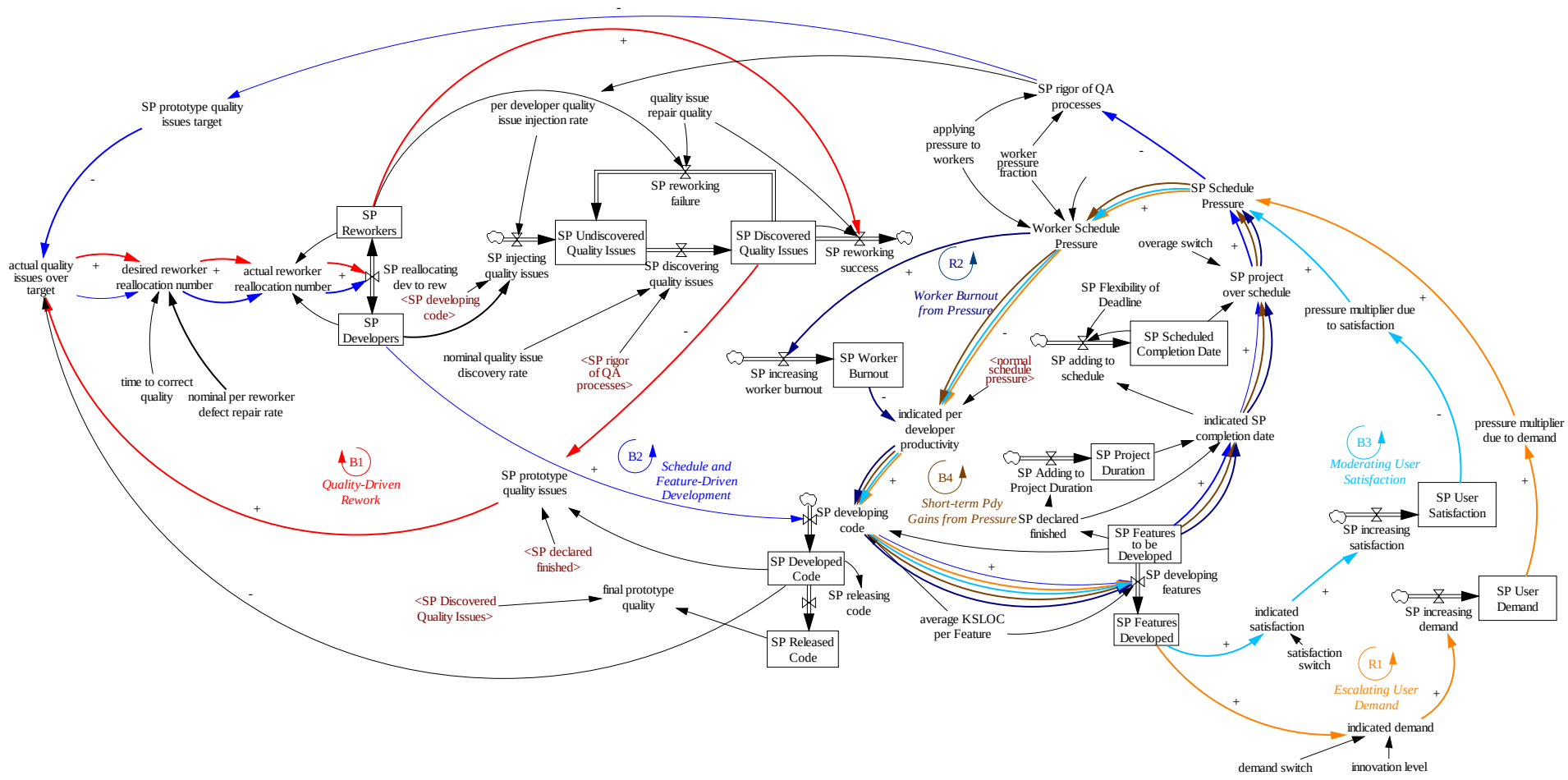
Throwing away the prototype results in better program performance

- However, very early transition or evolutionary development may also be viable



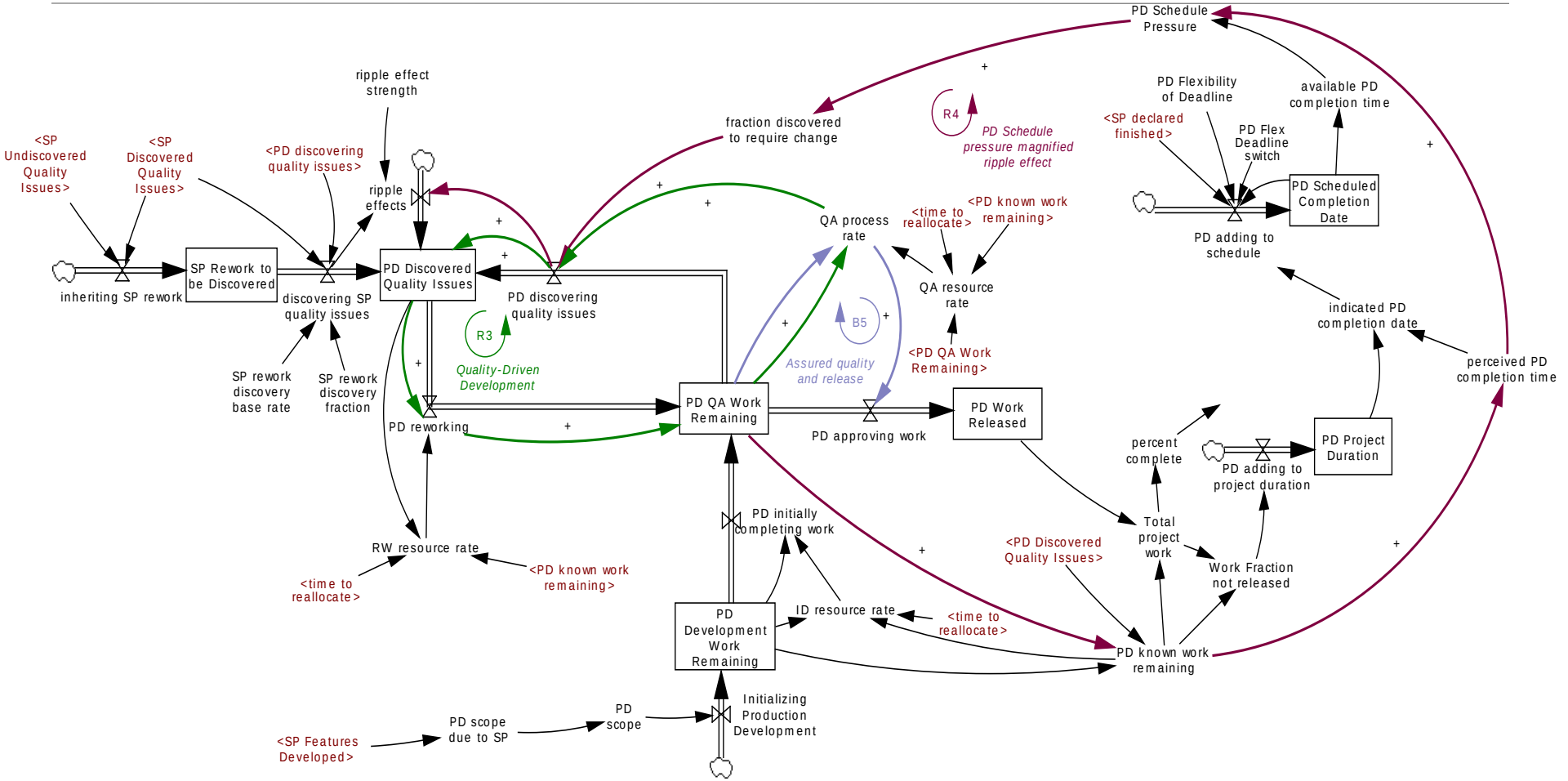
## The Evolution of a Science Project

# Full Simulation Model: The SP Sector



## The Evolution of a Science Project

# Full Simulation Model: The PD Sector



# Summary

# Summary

Key preliminary findings from "The Evolution of a Science Project":

- Undiscovered rework in a Science Project can lead to a "tipping point"
- Placing modest pressure on developers for limited periods shortens schedule
- The tipping point contributes to the "90% Done" syndrome
- The transition from 'science project' to production effort should be made early
- Throwing away the prototype results in better program performance

We can build on prior work in static models by developing interactive, executable models of key acquisition dynamics

- Turn existing software acquisition domain expertise into a more usable form
- Model complex dynamic interactions that we can't fully comprehend otherwise
- Good models produce key insights and raise important questions

The "I Already Knew That" effect

- Domain experts may say "I already knew that" about model results
- It's easier to point out something as obvious after it's been explained

# Learning Games for Acquisition

"Hear and forget;
See and remember;
Do and understand."

*—Chinese proverb*

# Why Learning Games?

## Inexperienced Acquisition Staff

- Acquisition staff often have inadequate experience in decision-making
- Well-intentioned decisions are undermined by adverse side-effects
- Poor acquisition management has major cost, schedule, and quality impacts

## Conventional Training is Limited

- Conventional training has been shown to be ineffective in preparing decision-makers for dynamically complex domains

## Learning by Doing

- Give acquisition staff a chance to learn how acquisition programs *really* behave, without risking an actual program

## Games and Simulations Teach Better

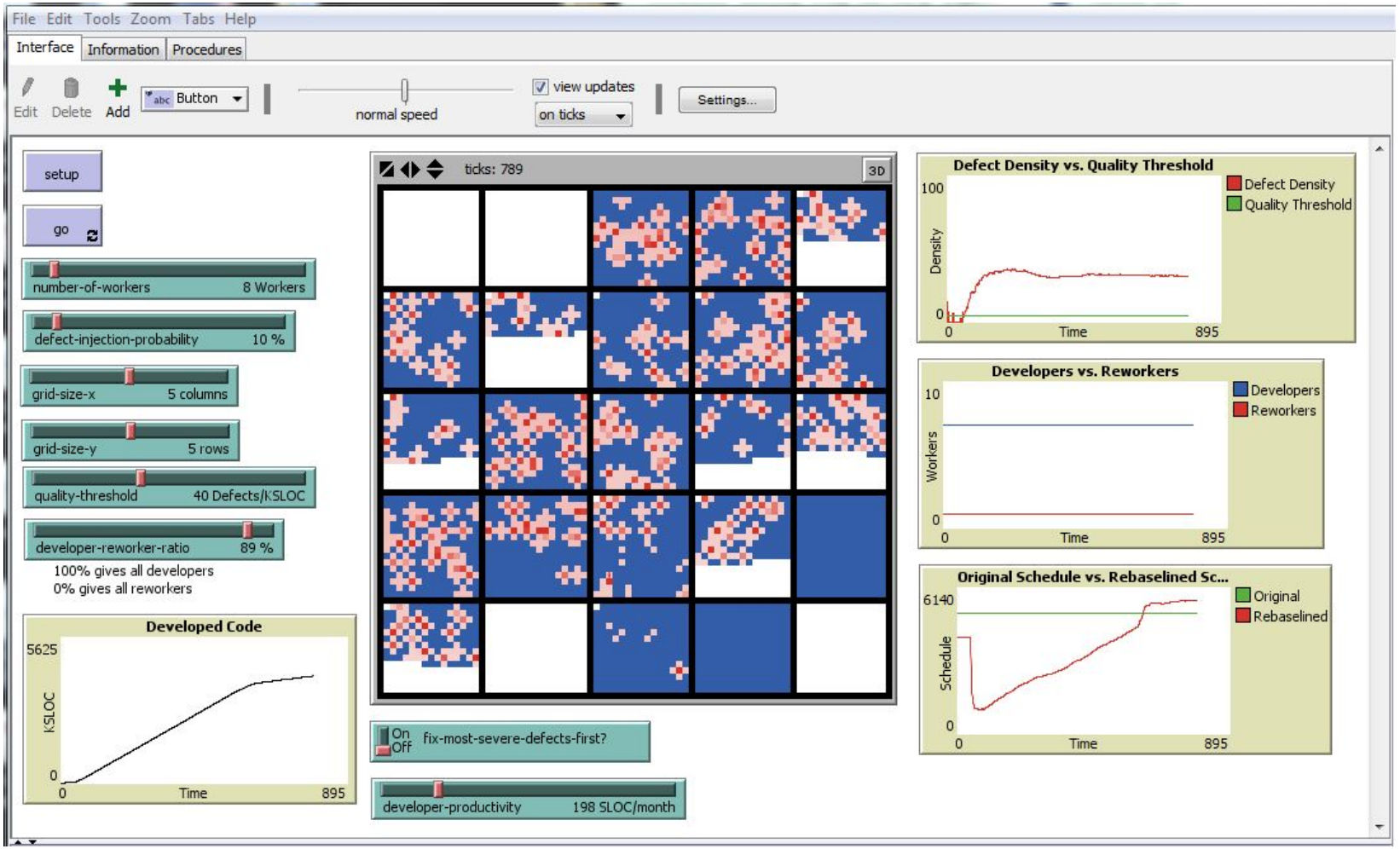
- [Cordova 1996, Ricci 1996] found that computer games and simulations enhance learning and understanding in complex domains
- "The hands-on learning model will be incredibly helpful to the DoD program offices" —*SEI Technology Forum attendee*

## Improved Learning Outcomes

- [Mayo 2007] found learning doubled for classes with interactive learning vs. only lecture

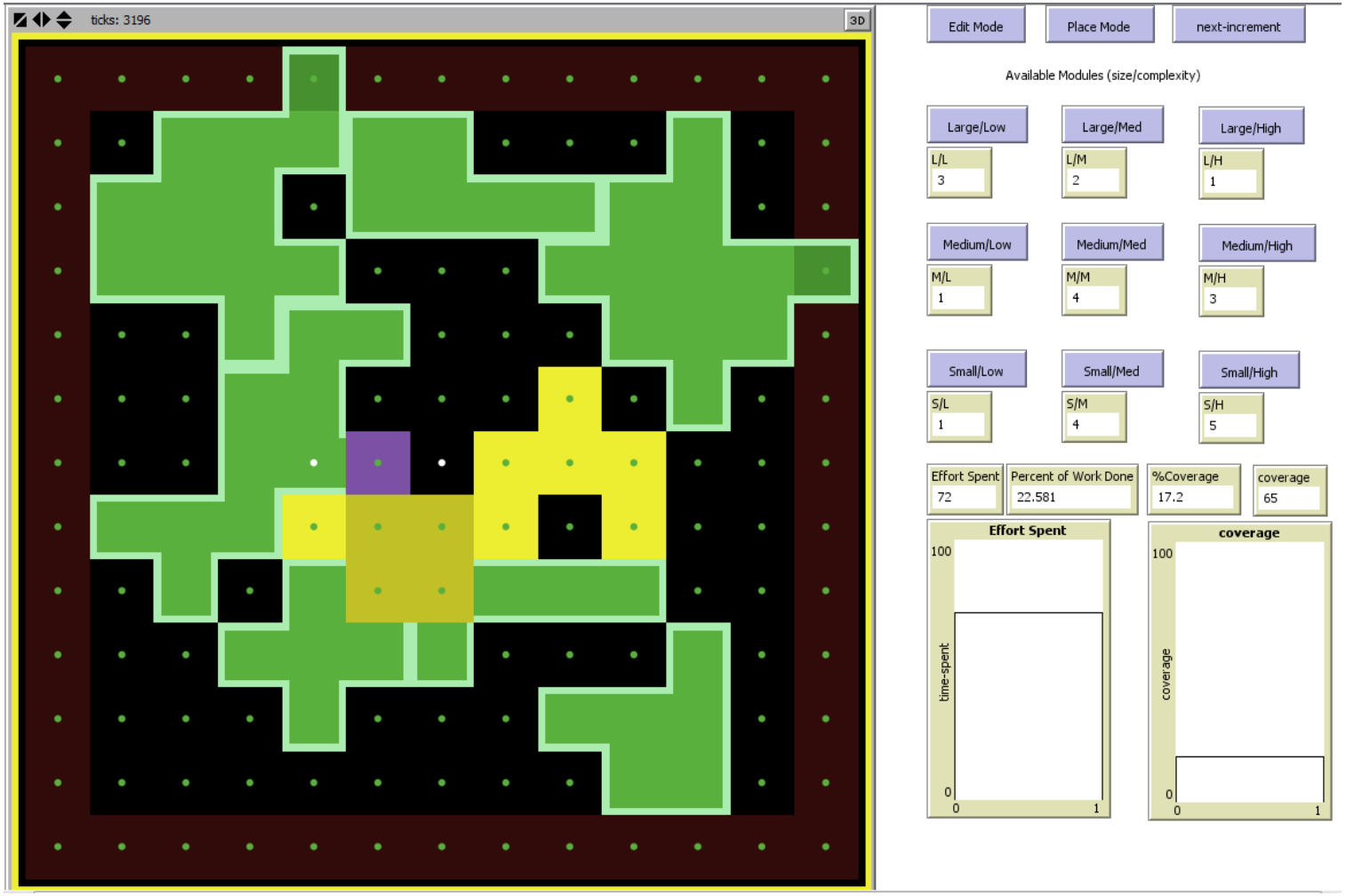
## Learning Games for Acquisition

# "Firefighting" Interactive Simulation



## Learning Games for Acquisition

# "Bow Wave Effect" Interactive Game



# Breaking the Pattern

"You cannot apply a technological solution to a sociological problem."
*—Edwards' Law*

"A clever person solves a problem. A wise person avoids it."
*—Albert Einstein*

# Managing the Acquisition Archetypes -1

If you're caught in a storm at sea, there may be little you can do to:

- Stop the storm, or even
- Get out of the storm

…But there *are*:

1. Things you can do *beforehand* to avoid it or minimize its impact, and
2. Things that you can do *during* the storm to help you weather it.

By showing the underlying *structure* of a dynamic, archetypes show where best to apply leverage to slow or stop it.

# Managing the Acquisition Archetypes -2

Once a recurring behavior has been characterized in a causal loop diagram, here are some key techniques for managing it:

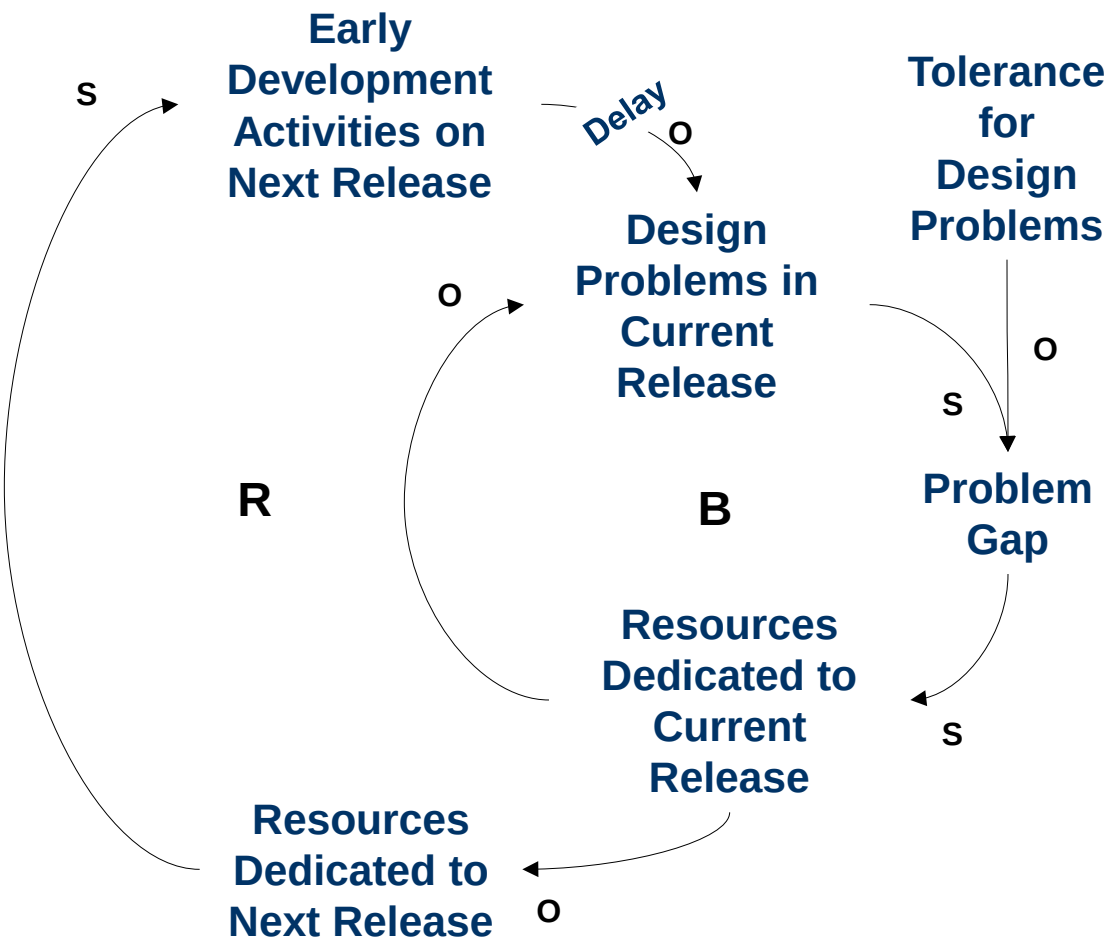
- Reverse the *direction* of the archetype
  - *Make negative dynamics positive ones by running them backwards*
- Slow down *unwanted* reinforcing loops
  - "*When you're in a hole, stop digging*"
- Accelerate *desirable* reinforcing loops
- Change the *limit* that a balancing loop is stabilizing around
  - *Change the equilibrium value to something more acceptable*
- Shorten the duration of a delay
  - *Make it easier to manage by making cause→effect more evident*
- Find *leverage points* where a small effort can have a large effect
- Look for *misaligned incentives* and try to align them

Each systems archetype has specific interventions for addressing it.

Knowing about these common dynamics is the best way to prevent them.

## Breaking the Pattern

# Managing "Firefighting"




*Fix*: Admit that diverting resources to fix bugs only fixes symptoms

*Fix*: Commit to fixing the real problem, with good estimates and more staff

*Fix*: Revise the plan/schedule

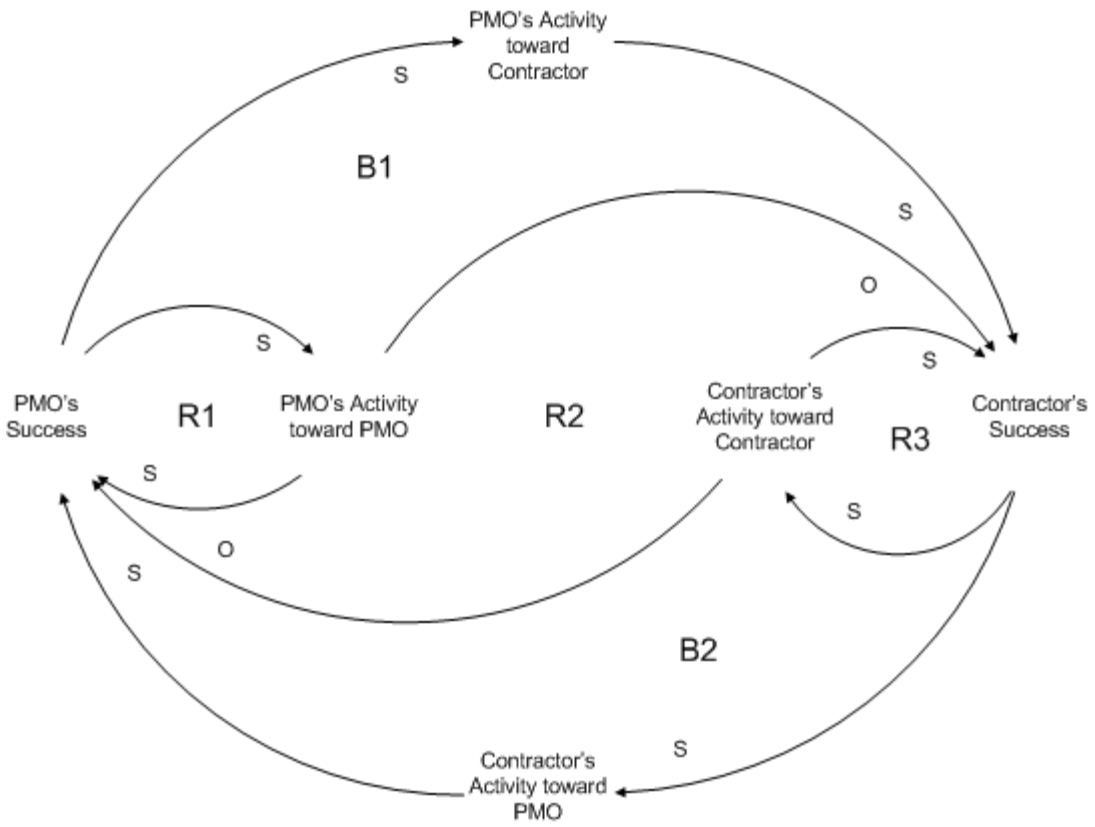
*Prevent*: Don't invest in new methods if you're already resource-constrained.

*Prevent*: Do resource planning across the entire project

*based on the "Fixes that Fail" systems archetype from "Past the Tipping Point: The Persistence of Firefighting in Product Development," Repenning, Goncalves, & Black, 2001.*

## Breaking the Pattern

# Managing "PMO vs. Contractor Hostility"




*Fix*: Break the escalation—one side must credibly commit to restoring trust

*Prevent*: Good communication is key—this dynamic only happens through poor communication

*Prevent*: PMO must "walk the talk" of "Trust, but verify"

*based on the "Accidental Adversaries" systems archetype*

# Solving Social Dilemmas

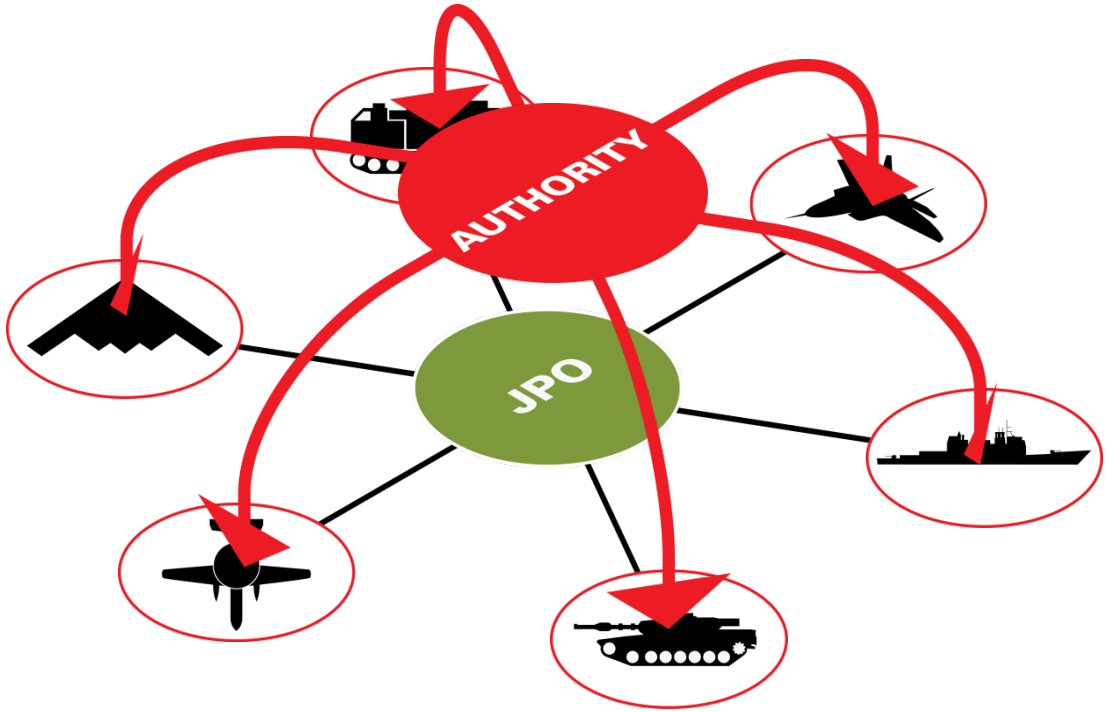
"Most [social traps]… are widely recognized to be genuine social problems, but the inclination to look at them as unrelated phenomena has obscured the possibility that similar sorts of solutions may be available across the board."

*—John Cross and Melvin Guyer, Social Traps*

# Solving Social Dilemmas -1

Resolving the “Tragedy of the Commons”:

- *Authority*: Designated authority regulates the good, restricts overusage
  - May be difficult and unpopular to enforce a mandate across services

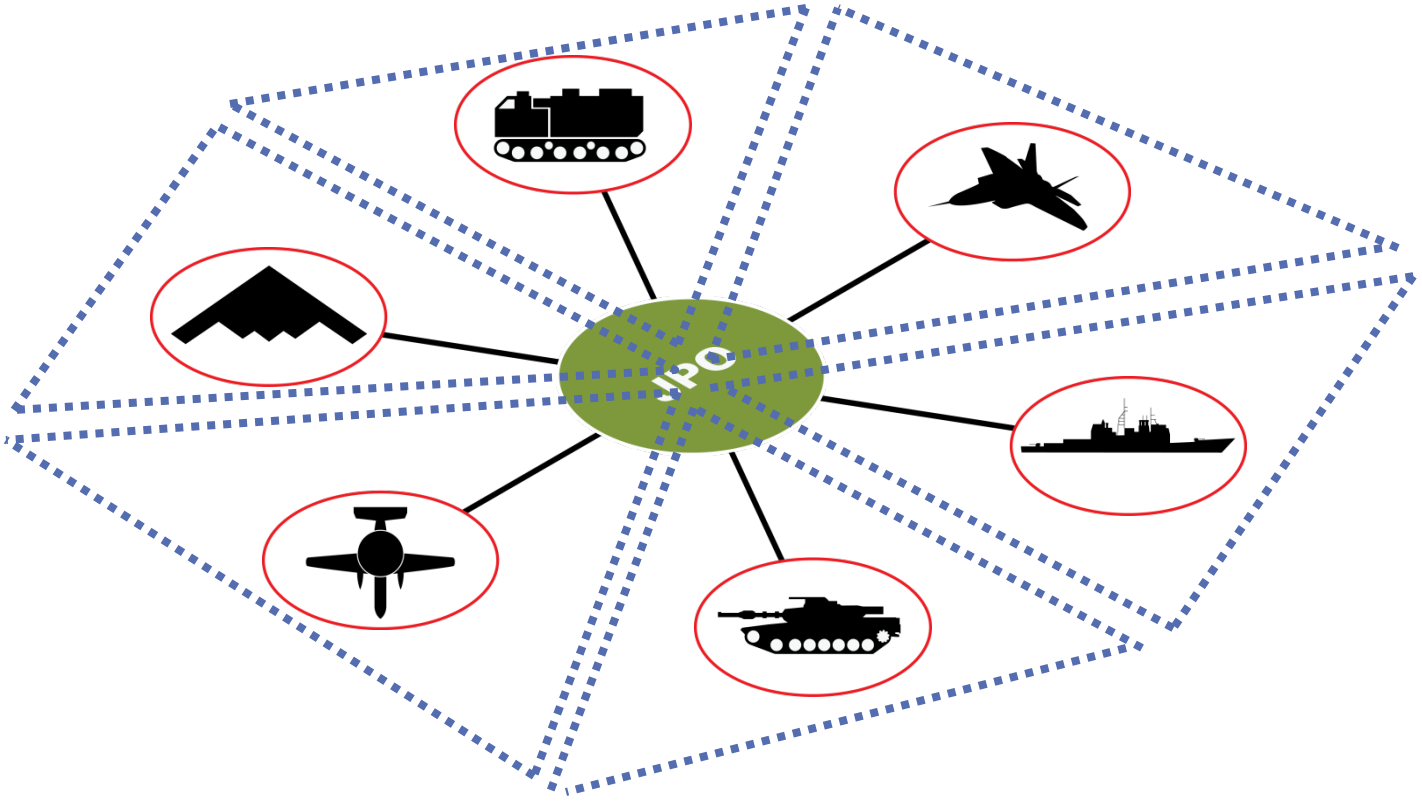



*from Cross and Guyer, Social Traps, University of Michigan Press, 1980.*

# Solving Social Dilemmas -2

Resolving the "Tragedy of the Commons":

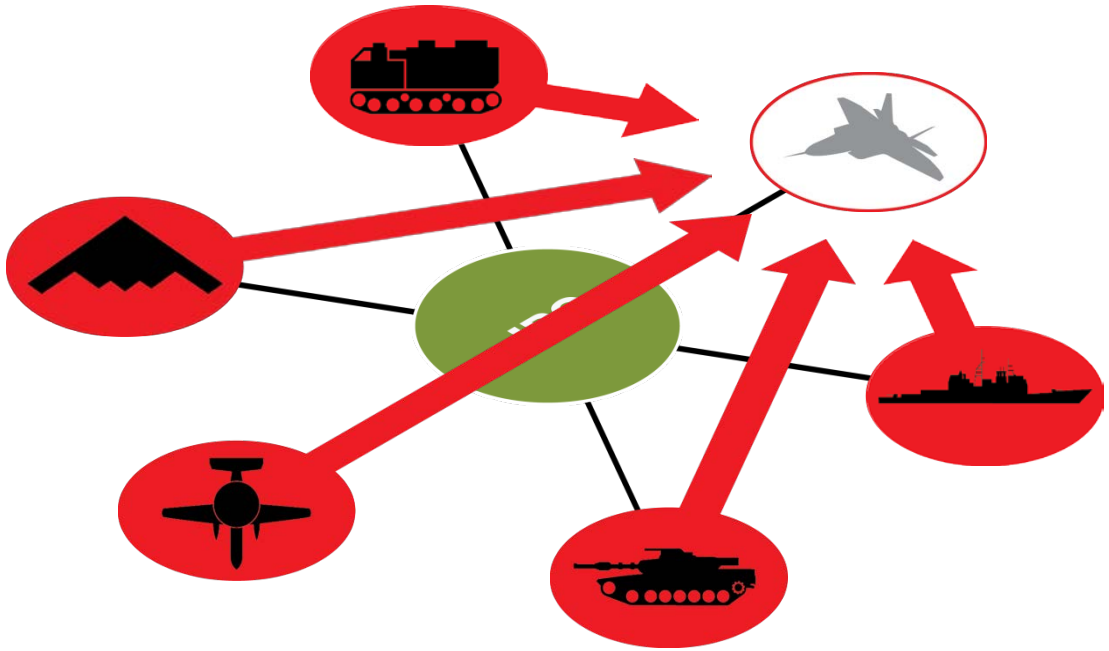
- *Privatization*: Converts shared ownership to private ownership
  - Each participant has a strong incentive to care for what they own
  - But privatization defeats the point of cooperation—causes "siloed" solutions



# Solving Social Dilemmas -3

Resolving the "Tragedy of the Commons":

- *Altruistic Punishment* [2]: Participants can penalize uncooperative partners
  - Significantly increases cooperation when used
  - Cost of using penalty discourages overuse, making it self-correcting



[2] *from Fehr and Gachter, "Altruistic Punishment in Humans," Nature, 2003*

# Solving Social Dilemmas -4

**Motivational**: Make people *want* to behave better

- ***Set Expectations/Reciprocity***: "If someone else does it, then I will, too"
- ***Awareness***: Raise awareness so that everyone knows how they should act
- ***Build Trust***: Let participants prove their trustworthiness so all are willing to cooperate
- ***Pulling Out***: Leaving the group if a partner defects sends a message to others

*from Cross and Guyer, Social Traps, University of Michigan Press, 1980.*

# Solving Social Dilemmas -5

**Strategic**: Give people some *reason* to behave better

- ***Reputation***: Build public reputations based on past performance to boost confidence
- ***Order***: Avoid a precedence order for using the resource, encouraging equality of use

*from Cross and Guyer, Social Traps, University of Michigan Press, 1980.*

# Solving Social Dilemmas -6

**Structural**: Change the rules so that people *must* behave better

- ***Authority***: Designate a leader/authority to regulate the use of the good
- ***Privatization***: Privatize the good so that each person pays for their use
- ***Rewards and Punishment***: Create clear rules, rewards, and penalties for behaviors
  - ***Reward the Group***: Reward people for group, rather than individual, success
  - ***Altruistic Punishment***: Allow participants to punish those who don't cooperate
- ***Assurance Contract***: Cooperate only if enough others also commit to do so
- ***Small Groups/Communities***: Small groups are more willing to "do it for their team"
- ***Exclusion Mechanism***: Find a way to exclude "free riders" from access
- ***Merging Free Riders***: Buy out the free riders so they have no incentive to free ride

*from Cross and Guyer, Social Traps, University of Michigan Press, 1980.*

# Sample Solutions to “Freeriding”

**Build the cost of the infrastructure into the cost of other things**

- ***Example*: Malls pay for restrooms and lighting by billing stores, who bill customers

**Keep people who don’t contribute to the infrastructure from using it**

- ***Example*: Satellite TV scrambles signals to exclude those who don’t pay
- ***Example*: Tollways

**Formalize ownership rights**

- ***Example*: Buffalo, whales neared extinction because they weren’t privately owned

**Have interested users pool their resources, and share results**

- ***Example*: Private research consortiums share results with their contributors

# Summary and Conclusions

"If we want to change people's behavior, then we have to create circumstances in which people are likely to act virtuously… If we think about reforming people's character, we're engaged in a futile pursuit."

*—Randy Cohen, "The Ethicist," New York Times*

# The Big Ideas -1

Lasting improvement to a system's behavior comes from changing the underlying system structure.

People are poor at controlling systems with large time delays between the cause and effect, because they obscure the connection between the two.

Diagrams of archetypes show the structure that lays beneath the visible problems, pointing out "leverage points" to help resolve them.

The ways people devise to *exploit* policies are *themselves* “emergent behaviors” that cannot be predicted from the rules of the system.

Understanding and changing the misaligned incentives at work beneath acquisition problems is key to improving program performance.

# The Big Ideas -2

Acquisitions fail primarily for non-technical reasons

- *Organizational, management, and cultural issues dominate*
- *"Technology has gotten ahead of our organizational and command capabilities in many cases"*

Misaligned incentives drive counter-productive behaviors

- *Programs put their own good ahead of other programs*
- *Programs put their good ahead of their service's good*
- *Programs place short-term considerations ahead of longer-term ones*

Understanding the problem is the first step toward solving it

- *If these problems were easy to solve, they wouldn't still be plaguing us*
- *There is no simple boilerplate answer—but there are solutions*

# The Big Ideas -3

Seemingly simple systems produce unexpectedly complex behaviors

- *New behaviors can emerge from interactions among components*

Small changes in initial inputs drive big changes in system results

- *Minor incidents can escalate into major catastrophes*
  - *PMO vs. Contractor Hostility (Accidental Adversaries)*
  - *Robbing Peter to Pay Paul (Success to the Successful)*

Assumptions contribute to failing acquisitions

- *Assumptions about others predispose you to behaving in certain ways*
- *Articulate underlying assumptions that contribute to misaligned incentives*

Lack of trust can degenerate into turf wars and a “death spiral”

- *If “individual/team gain” trumps the “program’s good,” bad outcomes result*

# For Additional Information

SEI Report: "*The Evolution of a Science Project: A Preliminary System Dynamics Model of a Recurring Software-Reliant Acquisition Behavior*"

SEI Report: "*Success in Acquisition: Using Archetypes to Beat the Odds*"

SEI Blog: "*Themes Across Acquisition Programs": Parts 1-4*

Website: http://www.sei.cmu.edu/acquisition/research/archetypes.cfm

Download all twelve:

- PMO vs. Contractor Hostility
- Underbidding the Contract
- Everything for Everybody
- The Bow Wave Effect
- Brooks' Law
- Firefighting
- "Happy Path" Testing
- Longer Begets Bigger
- Shooting the Messenger
- Feeding the Sacred Cow
- Staff Burnout and Turnover
- Robbing Peter to Pay Paul

# Joint Program Acquisition Experience Wanted!

We are analyzing the dynamic organizational behavior of joint and joint-interest programs as part of an ongoing research project.

We are conducting group modeling workshops to elicit key joint program behaviors, and are using the information to build a system dynamics model.

If you'd be interested in participating in a workshop, or collaborating with us in other ways, please contact:

William E. Novak
Senior Member of Engineering Staff
*Office*: 412.268.5519
*Email*: wen@sei.cmu.edu
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-3890

This material is based upon work supported by the U.S. 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.

Software Engineering Institute | Carnegie Mellon