

Identifying Acquisition Patterns of Failure Using Systems Archetypes

Finding the Root Causes of
Acquisition Problems

Bill Novak
Dr. Linda Levine
October 24, 2007



Purpose of this Presentation

To show how Systems Thinking and the Systems Archetypes can help to avoid common counter-productive behaviors in software acquisition and development programs

Agenda

- Systems Thinking
- Feedback Loops and Causal Loop Diagrams
- Selected Systems Archetypes
 - Fixes that Fail
 - Shifting the Burden
 - Limits to Growth
- Selected Software Acquisition and Development Archetypes
 - Sacrificing Quality
 - Firefighting
 - The Bow Wave Effect
- Seeing the Bigger Picture and Breaking the Pattern



Why is Software-Intensive Acquisition Hard?

Complex interactions between PMO, contractors, sponsors, and users

Limited visibility into progress and status—hard to comprehend

Significant delays exist between applying changes and seeing results

Unpredictable and unmanageable progress and results

Uncontrolled escalation of situations despite best management efforts

Linear partitioning (“Divide and conquer”) isn’t working well

Exponential growth of interactions as size grows linearly



Acquisition Programs are Dynamic Systems

Complex Interactions: Interactions between acquisition stakeholders are *non-linear*

Non-linear Behavior: Non-linear behavior defies traditional mathematical analysis because of the presence of feedback

Non-deterministic: Complex systems are not deterministic

Sensitivity to Initial Conditions: Results may vary greatly due to seemingly insignificant differences in the starting point(s)

Organizational: Key issues in software acquisition are management and organizational—*not* technical

Partitioning: Not possible with complex interactions between components



What is Systems Thinking?

Systems Thinking developed from work done by Jay W. Forrester at MIT while modelling electrical feedback effects

- 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 make things worse
- Improvement comes only from changing the underlying system structure



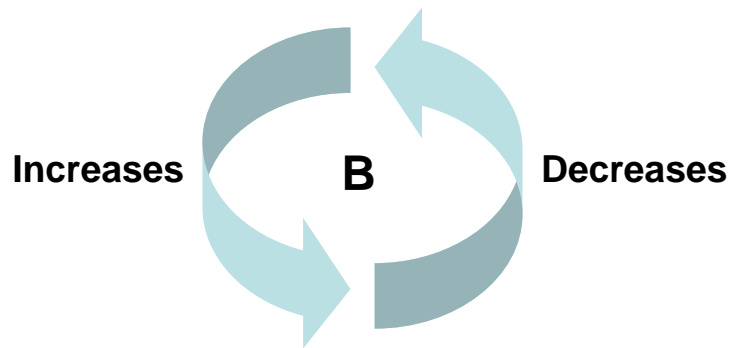
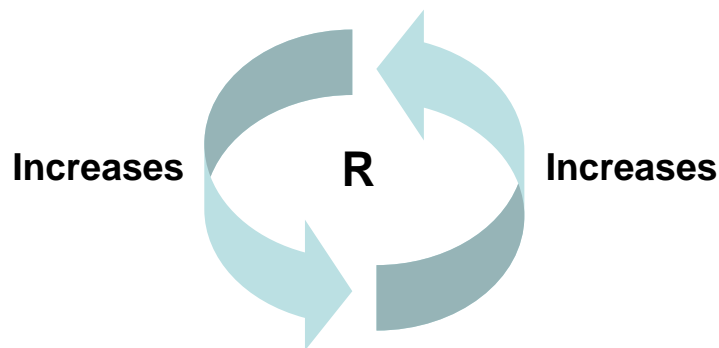
Causal Loop Diagrams (CLDs)

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

Show relationship direction by labelling them **Same (+)** or **Opposite (-)** to indicate how one variable behaves based on the previous variable

Consist primarily of two types of feedback loops:

- **Reinforcing** – Changes to variables *reinforce*, moving in one direction
- **Balancing** – Changes to variables *alternate*, achieving equilibrium



Time Delays

Much instability and 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 inherently poor at controlling systems with substantial time delays between cause and effect

Examples:

- Over-steering a large ship that is slow to respond, so it weaves back and forth
- A thermostat controlling a low-BTU air conditioner that’s slow to cool, so the house temperature bounces between too hot and too cold
- Inability to determine which surface, handshake, sneeze, or cough resulted in an infection



What are the Systems Archetypes?

The Systems Archetypes depict the underlying structures of a set of dynamic behaviors that occur in organizations throughout the world

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

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*)



Systems Archetypes

Over 10 recurring “systems archetypes” have been identified, including:

Fixes that Fail

- A quick fix for a problem has immediate positive results, but its unforeseen long-term consequences worsen the problem.

Shifting the Burden

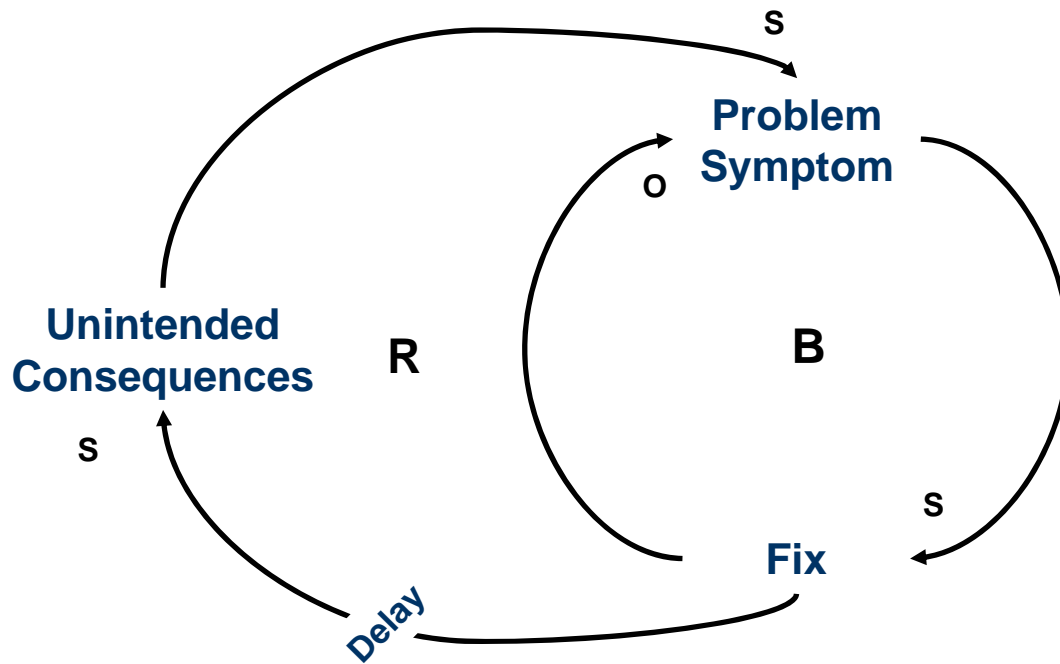
- An expedient solution temporarily solves a problem, but its repeated use makes it harder to use the fundamental solution.

Limits to Growth

- Initially rapid growth slows because of an inherent capacity limit in the system that worsens with growth.



“Fixes That Fail” – Systems Archetype

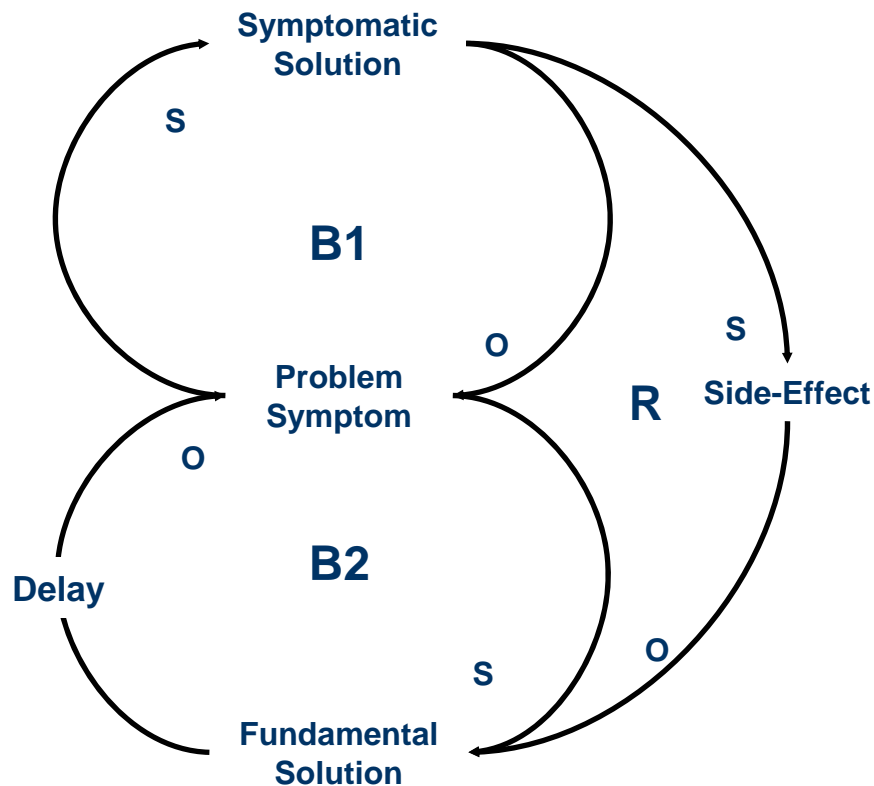


A quick *Fix* for a *Problem Symptom* has immediate positive results, but also has long-term *Unintended Consequences* that, after a *delay*, worsen the original *Problem Symptom* as the *Fix* is used more often.

based on “Fixes That Fail”



“Shifting the Burden” – Systems Archetype

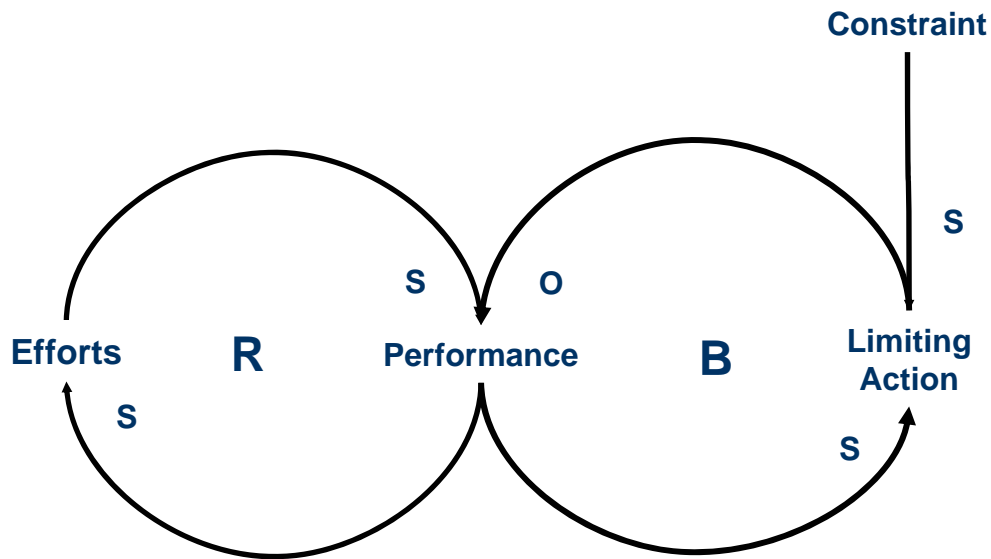


A *Symptomatic Solution* temporarily solves a *Problem Symptom*, which later recurs. Its repeated use over the longer term has *Side-Effects* that make it less and less feasible to use the more effective *Fundamental Solution*—trapping the organization into using only the *Symptomatic Solution*. Impatience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

Based on “Shifting the Burden”



“Limits to Growth” – Systems Archetype



Initially rapid growth slows because of an inherent capacity limit in the system that worsens with growth. As greater *Efforts* produce better *Performance*, there is a greater *Limiting Action* due to a *Constraint* in the environment, slowing *Performance*.

Based on “Shifting the Burden”



Acquisition Archetypes

There are many recurring patterns of behavior in software acquisition and development that have been modelled using Systems Archetypes and CLDs:

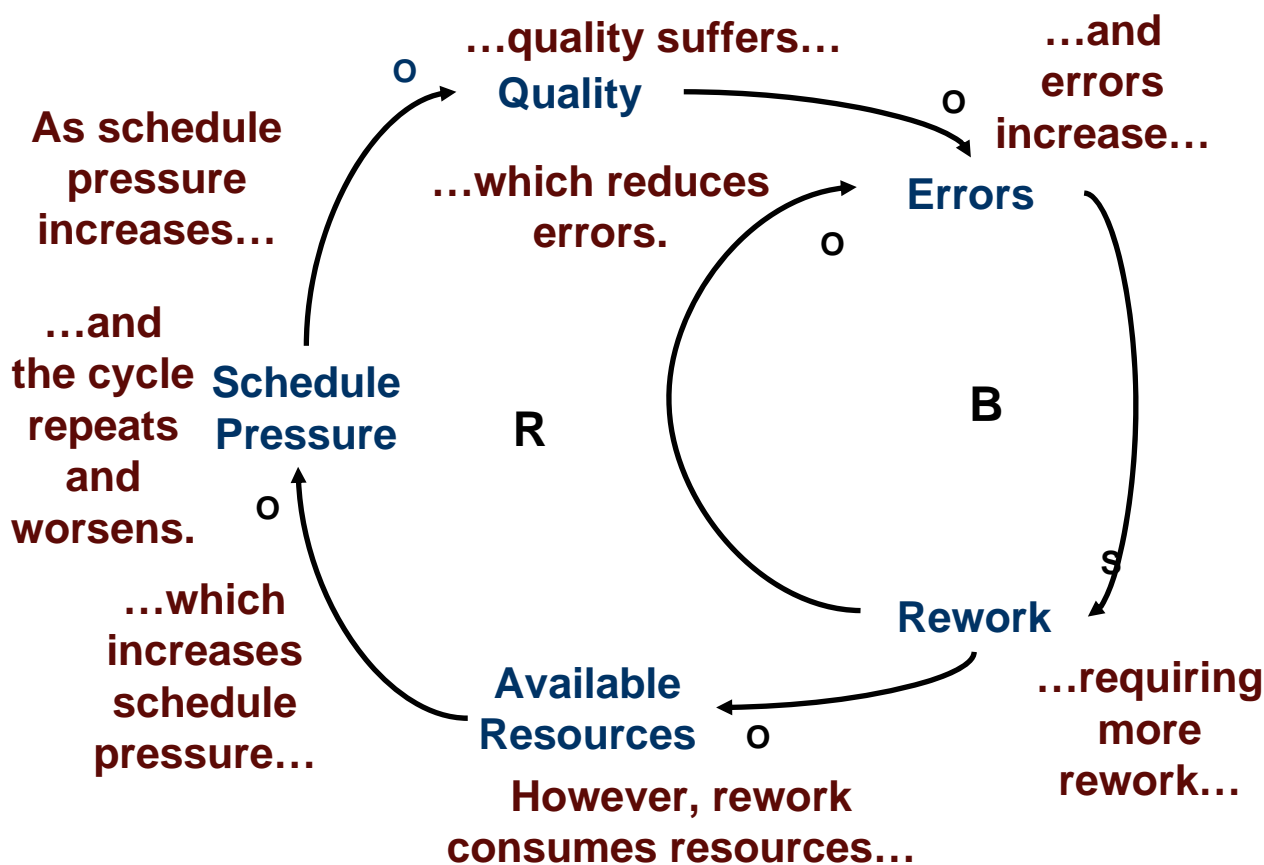
- Sacrificing Quality
- Firefighting
- The “Bow Wave” Effect
- Underbidding the Contract
- Shooting the Messenger
- Robbing Peter to Pay Paul
- Longer Begets Bigger
- The 90% Syndrome
- Requirements Scope Creep
- Feeding the Sacred Cow
- Brooks’ Law
- PMO vs. Contractor Hostility
- Staff Burnout and Turnover
- The Improvement Paradox

...

...



“Sacrificing Quality” – Acquisition Archetype

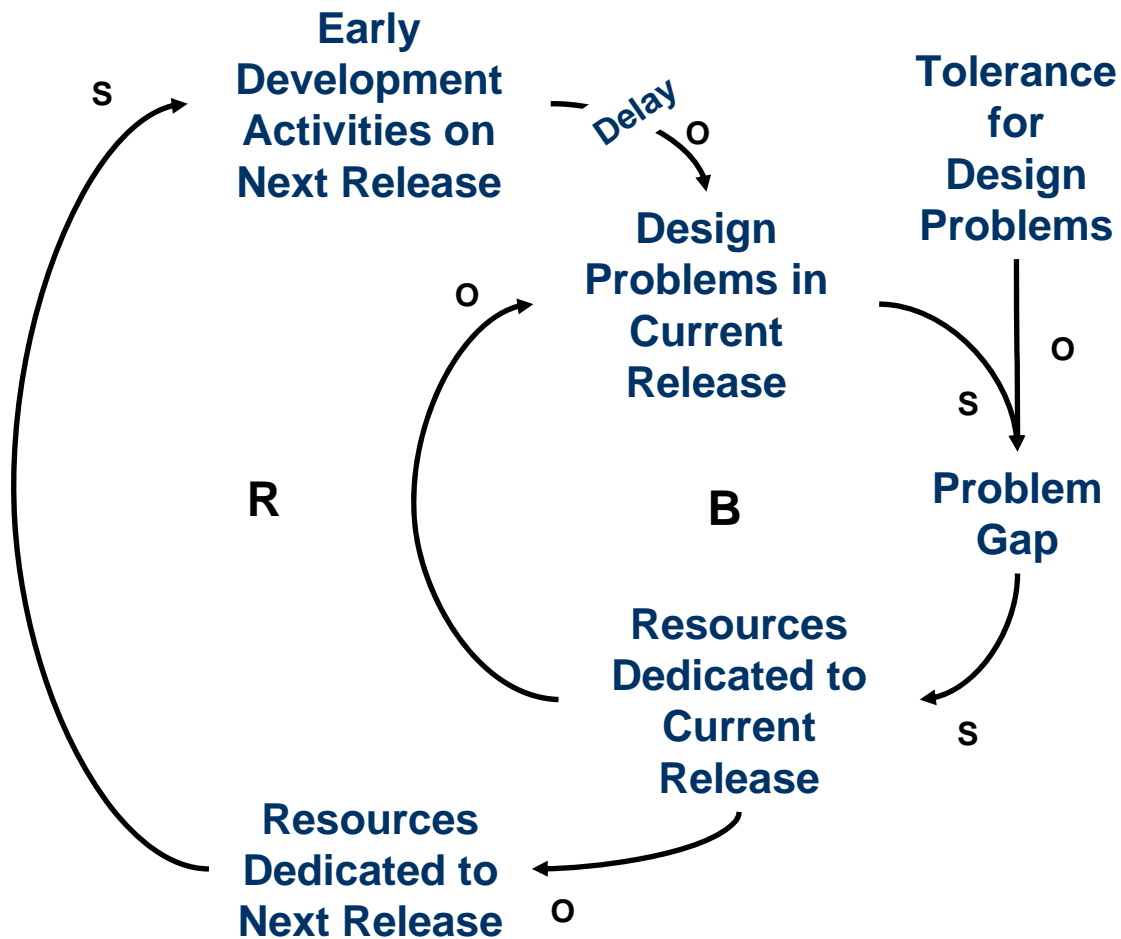


As schedule pressure increases, processes are shortcut, quality suffers, and errors increase—requiring more re-work. However, re-work consumes resources, which increases schedule pressure, and the cycle repeats and worsens.

based on “Fixes That Fail”



“Firefighting” – Acquisition Archetype

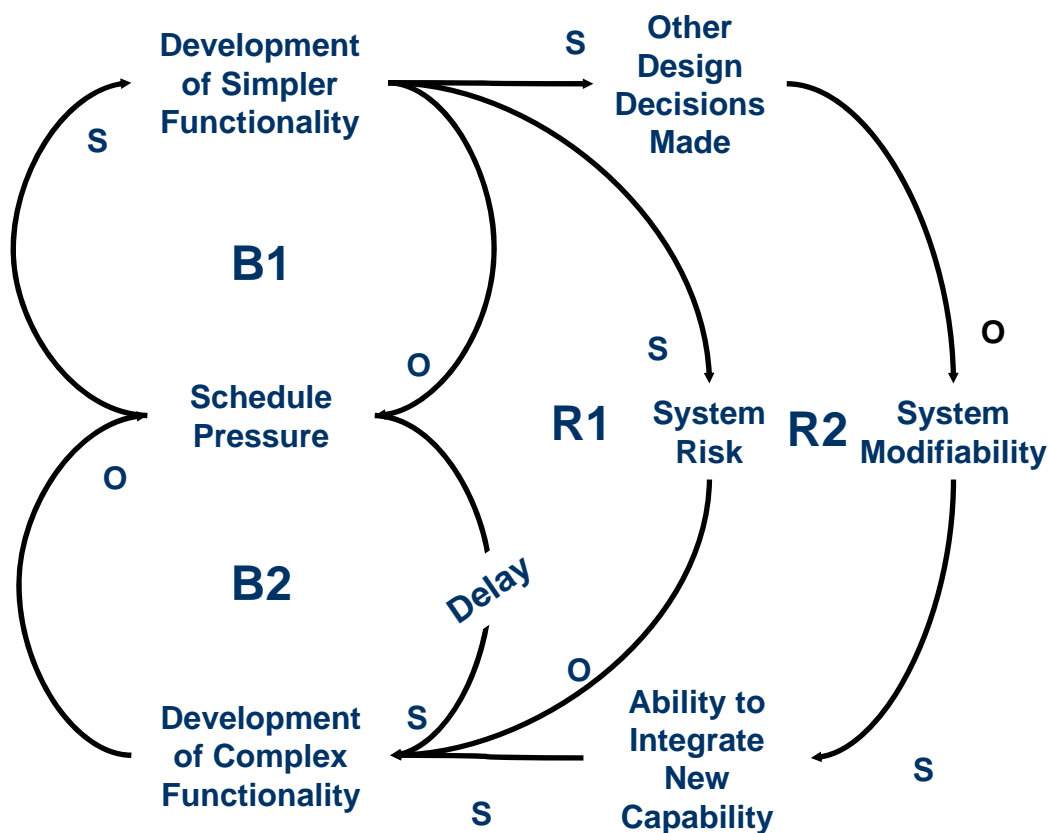


If a design problems in the current release are higher than the tolerance for them, more resources must be dedicated to fix them. This reduces problems, but now fewer resources can work on the *next* release. This undermines its early development activities which, after a delay, increases the number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

from “Past the Tipping Point”
based on “Fixes That Fail”



“Bow Wave Effect” – Acquisition Archetype



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 “Shifting the Burden”



The Bigger Picture/Breaking the Pattern

By showing the underlying structure of a dynamic, Causal Loop Diagrams show where best to apply leverage to slow or stop it—for example:

- Change negative dynamics into positive ones by running them backwards
- Slow the acceleration of unwanted reinforcing loops—“*When you’re in a hole, stop digging*”
- Change the limiting value a balancing loop approaches or oscillates around to something more acceptable.

Each systems archetype has specific interventions for addressing it

Knowing about the most common counter-productive dynamics is the best way to prevent them



Acquisition Archetype Concept Briefs

Acquisition Archetypes
Changing Counterproductive Behaviors in Real Acquisitions

Firefighting

When a project begins, no one intends to deliver it late, or to overrun their budget, or to give users a buggy system... it just happens. In fact, however, the total of overruns, late releases, and bugs is often considerable. Like most accidents, which means that they are due to coincidental causes rather than a single cause, but that those in charge of the program's troubles begin to see their contractor's estimates.

Growing Problems
With a poor estimation process, and trying to meet the government's desired schedule, the contractor underestimated the effort required in developing each release of the system. As the schedule slips, the contractor's estimates grow more and more unrealistic. The contractor's estimates grow more and more unrealistic. They end up releasing code at code reviews.

When system acceptance testing finally started they found that the current release had a very high failure rate in these cases, and the government felt:

"The contractor is taking resources off the next release to put them on current release firefighting, so there's a rippling effect."

All Hands on Deck
The contractor's solution was to "have those people working only on the most critical releases," so they called for all available staff to start firefighting. The contractor's solution was to "have those people working only on the most critical releases," so they called for all available staff to start firefighting. The contractor's solution was to "have those people working only on the most critical releases," so they called for all available staff to start firefighting.

How can I break this vicious cycle of schedule slips, cost overruns, high defect rates, and deferred functionality?

They had met their first system delivery deadline by deferring functionality, and they then checked to see the next delivery deadline because of their schedule work on it.

Acquisition Archetypes
Changing Counterproductive Behaviors in Real Acquisitions

The Bow Wave Effect

A Never-Ending Project
This is a true story—and one you've probably heard before. That's the point. It's about a pattern of failure, an archetype.

A government program needed to replace an aging COBOL mainframe business system—one so old that the costs of maintaining its obsolete hardware multiplied each year. The only people who could maintain it were now retiring, taking their knowledge with them. Yet the replacement project was stuck in low gear time dragged on, the focus of the program shifted, deadlines were missed. The sponsor's team member said, "...drifted, moved, and wadded, and done everything but die."

Finally, with the CIO under increasing political pressure to show IT results, the absolute, final deadline was set—just 18 months away.

How Bow Waves Begin
Could the development team get it done? Yes—but only if they kept to schedule and stuck to the project plan. And that's not what happened. Instead, the project almost from the start. First, educated guess (SWAG) estimation rather than measurement. [Requirements] were prioritized, and they got SWAGs, and they drew a line between available resources," including a team member. They approved [requirements] before they were coded. Some things moved from release to release if they fell below the priority line.

This practice of deferring sent ripples—bow waves—through the project. It doesn't seem malicious or even conscious by the project team. Quite the opposite. The effect was the end result of accumulated decisions that occurred right and expedient at the time.

The project managers didn't recognize the problem, or understand that the bow wave is, unfortunately, a common pattern in software development programs. Deferred and dropped functionality and system requirements project in a wave that washes up over delivery and project success.

Often, as in this project, the bow wave pass project teams in an impossible schedule squeeze.

"We don't compromise on quality, and can't add staff, so the only variable is scope—we just kept dropping functionality. But eventually records, and that meant we weren't allowed to convert them from the legacy delivery schedule got blown out of the water." A growing mass of work had to be done at the very end—when risk margin for further schedule slip.

Complexity Frosts The Wave
A number of errors fed the project's bow wave. Perhaps the most damaging one was failing to account for effects of complexity.

The team used a sequential development process, parcelling the system they were assembling, they built the initial processing module first (bundling the less complex records), and left the final modules (processing the most complex records) for last.

However, because no one really anticipated the complexity of the final modules during the planning phase, or the handling of the most complex records, the program didn't accurately estimate the feasibility or the effort of completing these tasks.

As you might have guessed by now, the team didn't meet its delivery deadline and at last report was still struggling with completing the final, most complex, processing module.

It was "a three-year program in its 13th year."

"We're trying to put 8 pounds of slop into a 5 pound bag."

SEI is producing a set of "Acquisition Archetype" concept briefs, analyzing recurring patterns in actual acquisition programs, and recommending interventions and preventative actions

Next Steps and Further Information

Extend the set of *Acquisition Archetypes*

- Eleven *Acquisition Archetypes* have been described to date
- Plan to identify additional acquisition dynamics and root causes

For additional information

- Visit the SEI website:
 - <http://www.sei.cmu.edu/programs/acquisition-support/pof-intro.html>
- Upcoming SEI Technical Note: “*Archetypal Patterns of Failure in the Acquisition and Development of Software-Intensive Systems*”
- Planned 2008 Workshop: “*Avoiding Failure in Software Acquisition*”





Software Engineering Institute

Carnegie Mellon



Software Engineering Institute

Carnegie Mellon