Identifying Acquisition Patterns of Failure Using Systems Archetypes

Finding the Root Causes of Acquisition Problems

April 2, 2008

Brian Gallagher

Challenges of Software

Software is invisible

- Typically we don't buy code – we buy a system

Software embodies "unlimited" complexity

- Complexity is often not understood nor appreciated

Doing it "right" requires persistence and discipline

- Not the most common traits throughout humanity
- Adopting good software engineering practices has a good ROI, but there is a "cash-flow" problem

Software and hardware technology continues to evolve very rapidly

Few program managers — the key decision-makers — have in-depth understanding of software technology

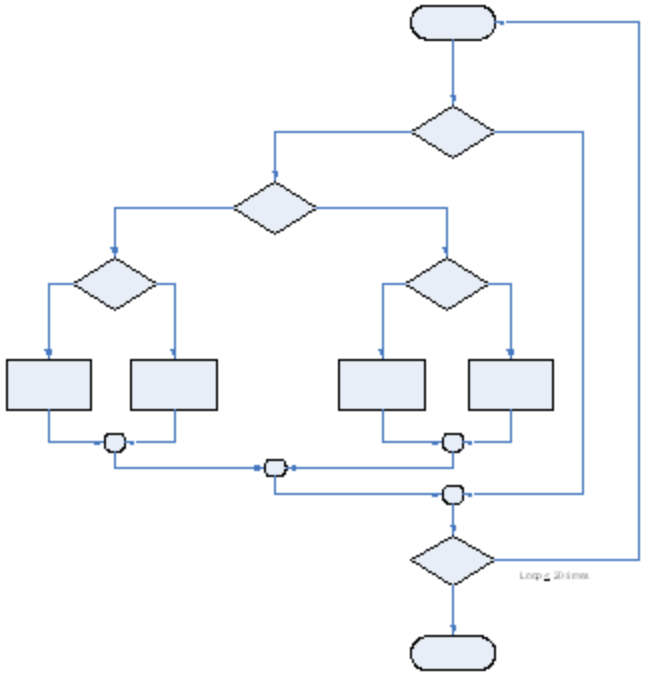
Software project management is still immature. Software engineering is arguably, still in its infancy

Challenges of Software

The flowchart might correspond to a 100 LOC module with a single loop that may be executed no more than 20 times.

There are approximately 10^{14} possible paths that may be executed!

For any but the smallest programs, complete path coverage for defect detection is impractical.



Adapted from Pressman, R.S., *Software Engineering: A Practitioner's Approach, Third Edition*, McGraw Hil, 1992

Challenges of Software

Typical Industry Software Quality at Delivery

- A 1,000 line-of-code (1 KLOC) program listing has about 20 pages of executable code
- For industrial software, typical *shipped* quality levels are 5 to 10 defects per KLOC or 1 defect in 2 – 4 pages
- A 1 million line-of-code (1 MLOC) printed listing stands roughly 5'7" and contains between 5,000 to 10,000 defects when *shipped*

For DoD acquisition programs, these realities are often ignored resulting in unrealistic schedules and unplanned test/fix cycles inserted to grow the reliability of low quality software.

The Buzzword Quagmire and Quest for the "Silver Bullet"

Open Systems

DoDAF

Interoperability

FEAF

ATAM

Acquisition Reform

Total System Performance Responsibility

Agile Acquisition

Time-Certain Development

Win-Win Spiral

Evolutionary Acquisition

Extreme Programming

Capability-Based Acquisition

Lean Six Sigma

CMMI

Team Software Process

Insight versus Oversight

Net-Centric Warfare

Service-Based Acquisition

Open Architecture

Service-Oriented Architecture

Incremental Commitment Model

Architecture-based Development

Earned-Value

Lean Acquisition

Systems Engineering Revitalization

Evolution of Concerns

Buzzwords come and go, the underlying concerns remain fairly constant

1. Software is developed by teams of between 5 and 20 people
   - A team can deliver ~XXX software lines of code in 6 months with highly predictable cost, performance, and quality (SEI's TSP, Agile Scrums, …)
   - Individual team performance can be extended to a team of teams – but breaks down in larger projects….
2. Optimizing team performance on larger project requires…
   - A software architecture that allows each team to operate autonomously
   - Disciplined project management and system engineering practices that facilitate communication across teams

Why is Software-Intensive Acquisition Hard?

Complex interactions between PMO, contractors, sponsors, and users
- Full chain of actions & their longer-term consequences are not clear
- 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 & effect
- *Example*: Steering an aircraft carrier

Unpredictable and unmanageable progress and results
- Limited visibility into real progress & status
- Complexity of interdependencies has unintended consequences

Uncontrolled escalation of situations despite best management efforts
- Misaligned goals 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 & address these interactions

Exponential growth of interactions as size grows linearly

What is Systems Thinking?

Systems Thinking is a method for analyzing complex systems

Developed by Jay W. Forrester at MIT 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 make things worse
- True improvement comes from changing the underlying system structure

What are the Acquisition Archetypes?

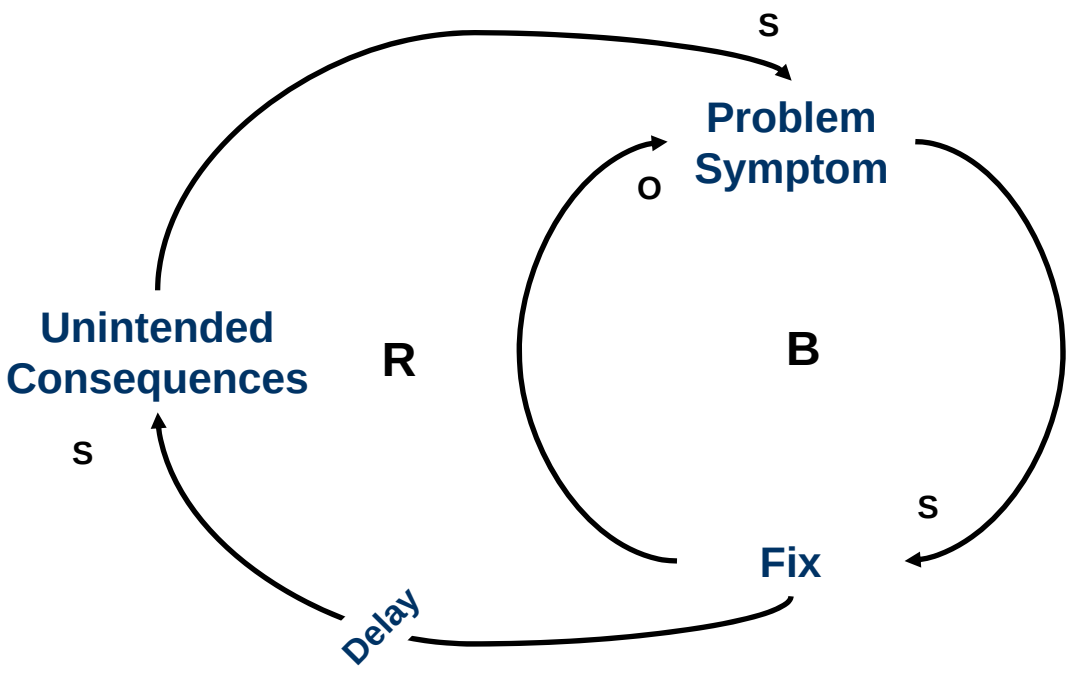
The Acquisition Archetypes depict the underlying structures of a set of dynamic behaviors that occur throughout acquisition organizations

- Each 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*)

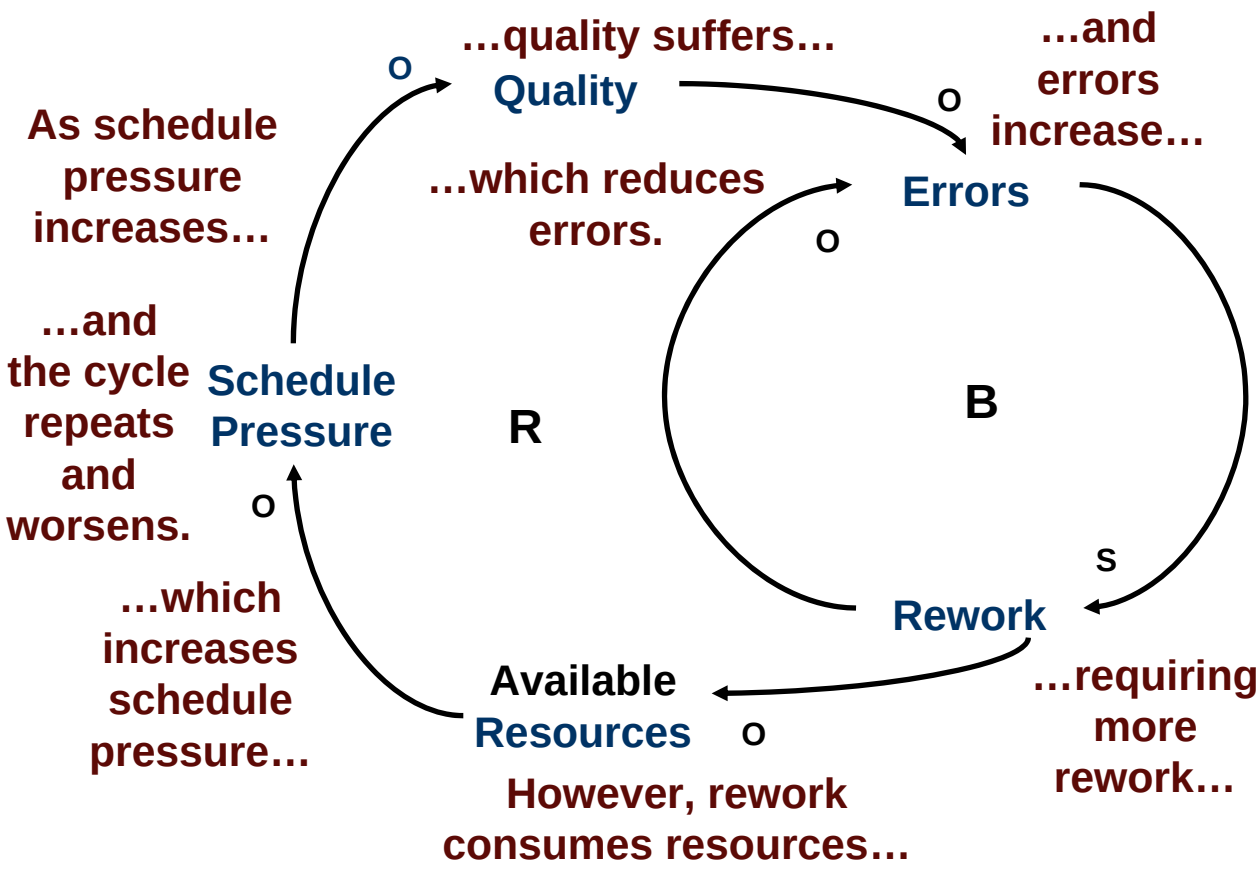
“Fixes That Fail” – Systems Archetype



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”

“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”

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

. . .

Acquisition Archetypes – Concept Briefs

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

Why Is This Approach Important?

Increasing complexity and acceleration in technical and organizational systems

Linear behaviors become nonlinear and unpredictable when combined

We lack problem solving methods that serve a “whole systems” view

Our current tools and methods are suited for handling ***detailed complexity***—where there are many variables.

Dynamic complexity refers to “situations where cause and effect are subtle, and where the effects over time of interventions are not obvious” (Senge, 1990, p. 71)

- When the same action has dramatically different effects in the short run and the long run
- When an action has one set of consequences locally and very different consequences in a different part of the system, there is dynamic complexity.
- When obvious interventions produce nonobvious consequences

Four Confounding Factors

1. Patterns & structural properties are hard to perceive & discern. Too much situational flux; few are looking closely or are in a position to look broadly.

2. Our problem solving strategies (for handling detail) are a poor match for handling dynamic complexity, and provide false assurance

- Few alternatives; this requires a radical shift in point of view and new problem solving methods

3. Work-life values can run contrary to a systems view—with a focus on short term, bottom line, and stovepipes. Actions based purely on these values often result in counter productive behavior. We think we are doing the *right* thing, but our perspective is too small or too short.

- Solutions that sound good but often backfire (insidious traps)
  - "results" focused
  - tyranny of consensus
  - low hanging fruit

4. Need to balance tackling the fundamental solution and achieving results. The challenge: Can you find "quick fixes" that contribute to the fundamental solution?

Next Steps and Future Directions

Pattern Library of *Acquisition Archetypes*

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

Collaborative Consulting

- Help customers identify program-specific, counter-productive behaviors

Learning Experiments

- Interactive “hands-on” exercises that demonstrate key dynamics in software acquisition programs

Acquisition Archetypes **Workshop,** 2 days, high interaction

- ***“Improving Acquisition Practice and Avoiding Patterns of Failure”*** will introduce the systems thinking approach, apply it to acquisition, present classic failure traps, and facilitate identifying specific failure patterns on their program

For More Information

Brian Gallagher

Director, Acquisition Support Program

Software Engineering Institute

4500 Fifth Ave.

Pittsburgh, PA 15213-3890

(412) 268-7157

bg@sei.cmu.edu

Army
Cecilia Albert, cca@sei.cmu.edu

Navy
Rick Barbour, reb@sei.cmu.edu

Air Force
John Foreman, jtf@sei.cmu.edu

Intelligence Community
Rita Creel, rc@sei.cmu.edu

Civil Agencies
Steve Palmquist, msp@sei.cmu.edu

Knowledge Integration & Transfer
Linda Levine, ll@sei.cmu.edu

http://www.sei.cmu.edu/programs/acquisition-support

Software Engineering Institute | Carnegie Mellon