Playing with Bulkheads, Layers, and Flows

A Strategic Analysis of Software Development, Complexity, and Control

Hans Konstapel, Leiden, January 1998

This chapter is drawn from "The Inevitable Cultural Revolution" (edited by H. Konstapel, E. Vreedenburgh, G.J.P. Rijntjes, SMO 1998)

Introduction

This article distills lessons from my experience managing complex software systems. By "management," I mean someone deliberately guiding a development process based on knowledge and understanding. By "software complex," I mean vast amounts of automated code built with extreme complexity—a condition that emerges as software ages.

By tracing developments in automation from 1970 to the present, I'll show how software diversity has exploded while individual control has declined. Today, we no longer speak of managing processes, but rather of autonomous growth.

I'll then examine human behavior in complex situations and demonstrate our fundamental weakness: **we are survival machines, not analysts.** Finally, I'll show how management becomes possible by playing with bulkheads, layers, and flows.

Part I: Three Decades of Software Evolution

Phase One: Programming (1970-1980)

The story begins with pure programming—the act of doing.

I first encountered computers in 1969 at Leiden University's computing center: a card reader surrounded by people carrying boxes of punch cards. It took a day to verify that everything you'd typed was error-free. And even then, the program might not work.

As mathematics students, we were trained in ALGOL and Assembler. ALGOL was designed to hide the incomprehensibility of Assembler, which involved shifting tiny binary data fragments between memory locations.

The power was intoxicating. After weeks of puzzling through problems and waiting for output, the program would finally work.

The Role of Language

Chomsky's theories profoundly influenced computer languages. By combining words and patterns (syntax), infinite possibilities emerge.

But I became fascinated with the reverse: finding words and patterns in results. **This is the translation problem—from one language to another—and we're still searching for ultimate solutions.**

Through studying the history of philosophy, I discovered that language and translation have preoccupied humans for centuries. Chomsky and Turing revealed something crucial: there's a vast difference between computer languages and human languages.

- **Computer languages** are unambiguous; meaning is clear when words are read sequentially
- **Human languages** require context-sensitivity; understanding often demands reading "back and forth" through the text

This difference has had enormous consequences for automation. Systems kept imposing computer language structure on users, making software increasingly complex when users refused to comply.

Phase Two: Managing Complexity (1980-1990s)

As software projects grew larger, management became necessary.

Projectmanagement was introduced, based on what became known as the "waterfall approach." Distinct phases were established—design, build, test, deploy—each with standards for documentation, training, and task descriptions.

But this assumed the world was rational and stable. Nobody realized how rapidly both automation and the external world would change.

The Database Revolution

Databases fundamentally transformed development methodology. The computer center shifted from a "paper factory" to a storage facility. The *given* was invented.

Architects emerged to establish spatial ordering between systems. Teams built enterprise-wide data models, hoping to capture all data's values, meanings, and descriptions. But **the assumption that data would remain stable in meaning and syntax proved false.** Data reflects how employees and customers see reality—and both perspectives change constantly.

The Pressure to Deliver

As preventive measures multiplied, many roles emerged around the builder: designers, quality controllers, testers, administrators, planners, trainers, and human resource managers. **The builder became surrounded by a multitude of non-productive managers.**

The irony: we still couldn't objectively measure what was actually productive.

Part II: Explosive Growth and Loss of Control (1990s)

The PC Revolution

The personal computer started a transformation where **individuals regained automation responsibility.** Software packages emerged. Networks connected everyone to everyone. Within weeks, a programmer could have their code running on millions of PCs.

Software stopped being designed—it began to *grow.*

The Scale Problem

The speed of change exceeded theoretical development time. By the time products launched, they were often obsolete.

Companies had to choose: make massive investments with uncertain returns, or follow the competition. **A new phenomenon emerged: projects that were completed but never deployed, because they were already outdated.** Fewer than 20% of software startups succeed.

Market-Driven Standards

Standards no longer came from institutions (like ISO), but from companies that captured large market shares. Microsoft's dominance exemplifies this—they specialized in market manipulation, pivoting entire project portfolios overnight when needed.

Part III: Understanding Human Behavior in Complex Systems

We Are Not Analysts

People construct explanations for phenomena, then live within those explanations. **After time, we think we "have it figured out" and narrate coherent stories about complex systems—without truly understanding them.**

Daniel Dennett's model is illuminating: consciousness resembles a group of friends playing a game where one person leaves the room, trying to guess a hidden pattern. The group answers randomly, but with a hidden rule: "Yes" for questions starting with A-M, "No" otherwise.

The departing person always finds a *meaningful pattern*—one the group never intended.

Similarly, **complex systems function as sources of random signals that our minds organize into coherent narratives.** We believe we understand when we've simply created a consistent story.

The Trap of Perspective

An analyst tries to understand their client's mental model. But they invariably map the *client's interpretation*—not the system itself—because the client is trapped within it.

How can someone at a distance from a process manage it? Management-by-walking-around occasionally works, but most managers remain in their offices, strengthening their distorted worldview through conversations with fellow managers. This creates the "black management hole."

Top-down development fails almost universally. The real world consists of loosely connected, overlapping structures. We cannot build comprehensive, unified systems.

When attempting to work downward from abstract principles to concrete details, we inevitably hit a layer structured differently, and we lose courage to adjust our explanatory principle. When we try to restore consistency moving upward, the same problem recurs. **We oscillate between levels of abstraction.**

Experienced developers stop trying. They begin in the middle, await the collision of misaligned pieces, and only then address problems. This is far more effective than top-down design.

Ideas Control People

Ideas—memes—drive humans, just as genes do.

Some ideas are more powerful than others. Denett based the term "meme" on "gene" to suggest that comparable evolutionary principles operate at the level of concepts. **Concepts develop like species, without human intervention—humans merely serve as transformers.**

Excellent developers don't think; they trust intuition. They have a direct link between idea generation and program creation. After the program exists, they're finally able to *explain* their method.

We must give people space to fail. Our brains protect us by making us feel in control, sometimes by distorting truth. Managing people must recognize how memes (not just rationality) guide human behavior.

Part IV: Evolution Theory Applied to Software

Understanding Evolution

An evolutionary process requires a universe (the real world, a computer, a brain), a substance acted upon (matter, ideas), and two mechanisms: **variation** (random change) and **selection** (what survives).

Nature displays layers with varying degrees of evolutionary exposure:

- Outer layers change rapidly
- Inner layers change slowly and are more stable
- Inner structures protect outer layers

Layers communicate with each other. An outer layer essentially becomes the "universe" for the inner layer.

Kauffman's Insight

Richard Kauffman discovered that **order is an intrinsic property of certain self-replicating networks.** The critical factor is the relationship between connections, nodes, and feedback.

Networks with approximately two connections per node exhibit high order naturally. Networks with more connections show lower order. Networks can oscillate between stability (solid), fluidity (liquid), and chaos (gas).

Most adaptive systems operate at the edge of chaos—the liquid state where structures can rapidly reconfigure.

Applied to software: as we connect more components (PC networks, the internet), we increase diversity but risk tipping into chaos. **The velocity and extent of planning affects network growth.** Ad-hoc work produces strong order but eventual stagnation; over-planning generates excessive diversity and chaos.

Kauffman's crucial insights:

- **More connections = more diversity** (explaining software explosion via the PC)
- **Further future planning = more diversity** (suggesting careful time-management)
- **Better design = greater fragility** (well-designed systems break catastrophically under change)
- **Disturbances always occur** (Murphy's Law has patterns)
- **Better models of reality → chaotic behavior** (perfect information systems can collapse organizations)

Part V: Networks, Bulkheads, Layers, and Flows

The Network as Central Concept

Networks have become crucial in automation, biology, and cognitive psychology. Though nodes vary in content (computers, genes, neurons), similar conclusions emerge.

Information flows through networks via a medium (electrons, RNA). **Structure emerges through deliberate management of connections and feedback.**

By placing or removing "bulkheads" between compartments, we determine whether regions develop independently or together. Information then flows through different channels. Some paths lead nowhere; others create loops. Coupled sections seek equilibrium; when they can't achieve it, patterns become chaotic.

The most flexible structure is fluid (liquid): some regions structured, others chaotic, constantly reconfiguring.

Human Networks and Organizations

When we form networks of people and their communication, we create organizations and projects —patterns that persist.

These endure through self-reproduction and resistance to change. **The network structure itself enables this persistence.**

With communication networks, increasing connections create new stable patterns—"network organizations." Hierarchical barriers between departments cease working; the "top" becomes isolated from lower levels.

A fluid human structure is one approaching dissolution. If a team develops a fixed pattern, managers must introduce new "looseness" to prevent stagnation. Yet constant disruption creates stress.

Humans share concepts. Stable structures mean stable data-exchange patterns and shared concepts. **Concepts change people; people change concepts.** More connections → more shared concepts → equilibrium.

But when external reality diverges too much from internal reality, someone must disrupt equilibrium. Humans naturally adjust their internal image to match their internal world—not the reverse.

Concept Networks

In Douglas Hofstadter's *Fluid Concepts and Creative Analogies*, creativity emerges from "subcognitive pressure" that probabilistically influences representations.

Creativity appears through errors in stable concept structures—"slips of the tongue"—that reveal new combinations. These occur under stress, often spontaneously, after periods of tension. Artists and innovators leverage this deliberately.

Some people generate brilliant ideas constantly; they combine disparate domains. These individuals are often polymath geniuses, linking previously disconnected fields.

Realization Through Patterns

Christopher Alexander spent fourteen years on *The Timeless Way of Building*, exploring how three-dimensional structures (gardens, buildings, cities) achieve "Quality Without a Name"—that feeling that something *fits*.

This arises through "design patterns"—approaches to balancing a force-field in a particular context. These forces include natural ones (gravity) and human ones (the need for solitude balanced against the need for community).

Alexander's insights:

- Patterns overlap, creating new patterns
- A pattern can initiate others
- Good pattern languages are "morphologically complete"—you can visualize the end result
- Patterns must balance forces across all possible combinations
- A culture is a specific collection of patterns

Software development has adopted design patterns, recognizing that complex systems follow recognizable, repeating structures. Just as Alexander observed that architecture follows patterns, software architecture naturally develops them.

Part VI: Power, Language, and Memes

Powerful Concepts Possess "Quality Without a Name"

Memes—concept structures—function as force-fields in balance. Usually combinations of opposites.

Powerful memes have "quality without a name." Mathematics is perhaps the most powerful coherent meme on Earth—capable of serving as the foundation for almost everything.

Concept complexes function as selection mechanisms for other complexes, as lions force gazelles to evolve faster. **Strong memes suppress weak ones.**

Strong memes likely possess self-reference and multiple feedback loops in their semantic network. They reach horizontally across many structures and create vertical closure.

Mathematics is the purest, most coherent conceptual world. A good proof gives that feeling of rightness. The mathematical complex is so large and powerful that it resists virtually all competing conceptual structures.

We may eventually need to adjust reality to match mathematics, not the reverse—the pressure to realize is immense.

Part VII: Synthesis and Conclusion

Managing the Unmanageable

Controlling a global network of people and memes appears impossible. The current flows according to its own logic.

Islands of stability sometimes emerge where we can attempt to provide direction.

This direction comes through moving bulkheads—isolating and reopening flows.

We manage by placing or removing barriers between compartments, controlling which regions develop together. Some regions must remain fluid; others need temporary structure.

Where It All Goes

Where everything goes remains a mystery.

We must be prepared for this uncertainty.

I'm betting on mathematics as a last resort—probably because I'm a mathematician, trapped in my own conceptual world. And so I discover, repeatedly, how memes keep us engaged.

This article has been a struggle with time and concepts. Several insights emerged only under the final pressure—confirming the theory itself: ideas precede thought, and profound understanding often arrives only in constraint.

Perhaps these patterns will resonate with you and inspire new directions.

Or perhaps I've been incomprehensible and tedious.

The memes continue their work regardless.

Key Themes Summary

Theme	Core Insight
Language & Complexity	Computer languages impose their logic; human adaptation makes systems increasingly complex

Human	We construct narratives for complex systems rather than understanding them
Evolution &	Software develops like biological systems—through variation and selection
Networks	Order emerges naturally in systems with moderate connectivity; too much
Pattern	Complex systems reliably develop recurring patterns; these embody "quality"
Management	Control comes not from top-down planning, but from managing bulkheads and flows within networks
Time Horizons	How far ahead we plan directly affects system diversity and adaptability
Memes	Ideas—not just rationality—drive human and organizational behavior