Principled System Architecture

prerequisite for resilience

Robert Rasmussen

Copyright 2012 California Institute of Technology. Government sponsorship acknowledged.

"Resilience"

- Literally, the ability to spring back
  - Resilient systems work, no matter what
- Brittle systems are not resilient
  - Small problems easily break them

Engineered Resilience

- Resilience in nature arises over many generations through trial and error

- Engineered resilience must often be right the first time

Many Ways to Fail

- Stakeholder concerns that aren't properly appreciated, reconciled, or accommodated
- Progress thwarted by intolerance to development uncertainties
- System interactions that come as a surprise
- Late discovery of design or implementation errors
- Unvalidated assumptions
- Poor risk assessments
- Inadequate or misapplied V&V
- Unethical conduct
- Flight manifestation of uncorrected design flaws
- Fatal defects in materials, implementation, workmanship, tools…
- Unusual or unanticipated environments
  - Stress damages the system
  - Control outside the validated regime
- Inability to degrade gracefully
- Changes in mission or usage that violate assumptions
- Operator error
- Malicious action

et cetera!

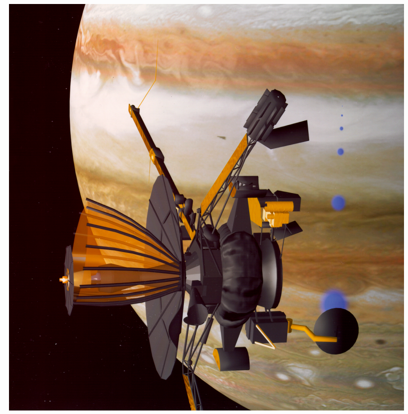
Often an unfortunate combination of things
Often resulting in convoluted behavior

We Know
What Resilience Looks Like

Apollo 13*

Innovative repurposing

Galileo*



Computing margin and flexible re-programmability

* *So far, dependent on many clever* ***people*** *and considerable* ***luck***

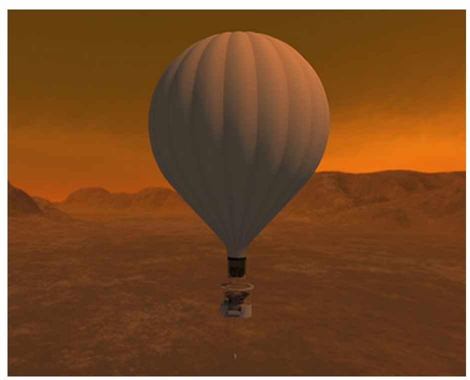
Hubble Space Telescope*

On-orbit instrument replaceability

T-800

Graceful degradation and goal-oriented behavior

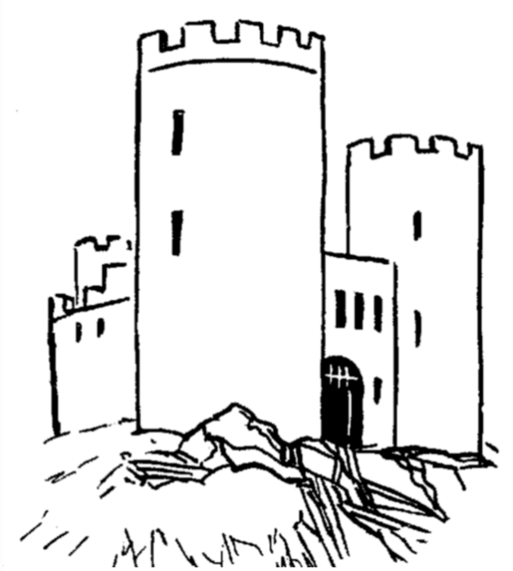
Titan Balloon



Self-direction and tolerance for variety

Still Largely a Defensive Exercise

- Robust engineering tolerance is largely concerned with prescribed variation
  - Depends on an assured perimeter
    - Qualification ranges, diligent oversight, “test as you fly…”, conservative analysis…
  - And ample resources
    - Overdesign, operating margins, redundancy, schedule slack, opportunity to retry…

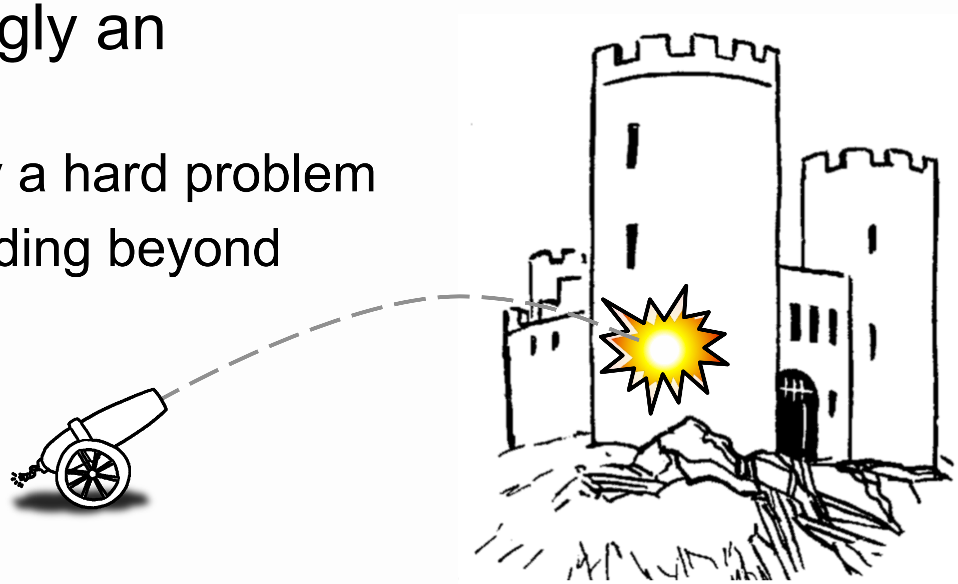


Robustness is like siege defense: Strong walls and plenty of supplies, but not much freedom

- Okay for lots of systems, but always a limiting strategy
  - Retry or retreat can’t be the answer to every challenging situation

Do We Defend or Adapt?

- Defense is increasingly an incomplete strategy
  - Robustness is already a hard problem
  - But problems are trending beyond robustness to matters of astuteness



- Defense must be augmented with Adaptation
  - Figure out what's happening and deal with it creatively
  - Less canned responses; more cognitive, coherent deliberation
  - Depends on acquiring knowledge, and an ability to solve problems and to improvise
- This makes a hard problem *much* harder

Tough Architectural Questions

- When is resilience the right answer?
- Where does resilience fit among all other system concerns?
- What are the technical and programmatic building blocks of resilience?
- How does one provide a fundamental, reasoned basis for declaring that a system has resilience?

A Systems Engineering Challenge

- No simple sum of technologies will do
  - Resilience of a system can’t be derived from resilience of its parts
  - Resilience can’t injected into a system or added onto it
- Like all architectural considerations, resilience is a *system* characteristic
  - Simple problems can topple whole systems
  - All parts of system must participate in solutions
  - Adaptation requires reasoning about the system
  - Reasoning requires understandable systems
- “The System” is not one thing, but many
  - Variation, surprise, and invention are to be expected, not avoided
  - Adaptation solutions are open ended
  - Engineering the design space is “architecture”

A Definition

- A **System** is anything greater than the sum of its parts
  - Every part affects others — the parts become one
- *New* attributes, not intrinsic to the parts, arise *solely* from these **interactions**
  - This phenomenon is commonly referred to as ***emergence***
- Systems are intrinsically about
  what is added
  through interaction

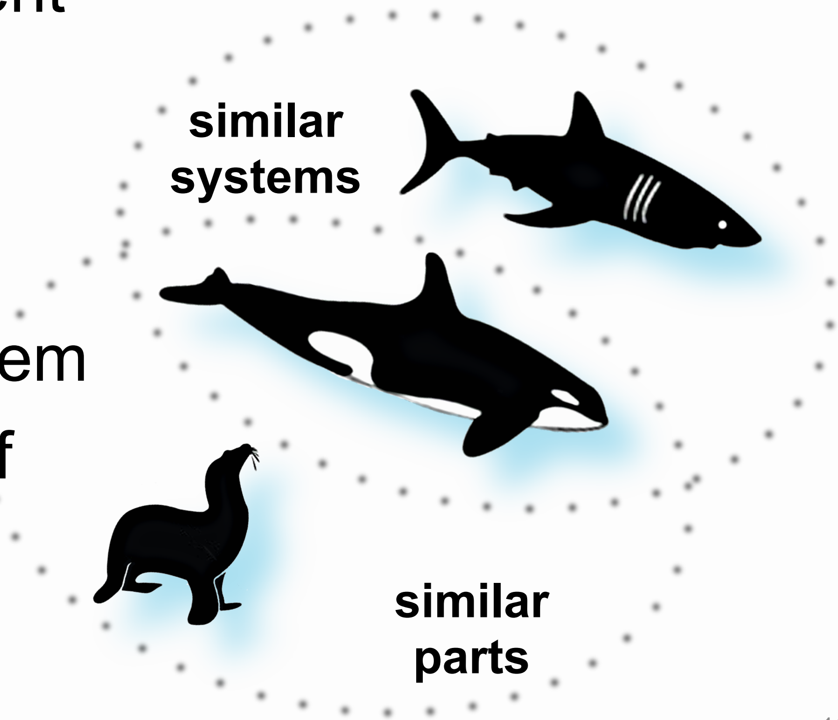
Interaction, Not Interface

- Interactions can be…
  - Exchanges of material, energy, or information
  - Coupled attributes or shared constraints
  - Planned or not planned
- Interfaces *per se* are not paramount
- What matters is how each part ***affects*** the others

Winslow Homer (1836 –1910)

Emergence, Not Integration

- Additions can be new capabilities, functions, or behaviors . . . **abstract entities, but…**
- **The resulting systems are new, *real things* in their own right**
  - *Not merely* an arrangement of parts and interfaces
  - Similar arrangements of different parts can yield essentially the same system
  - Different arrangements of similar parts can yield quite different systems

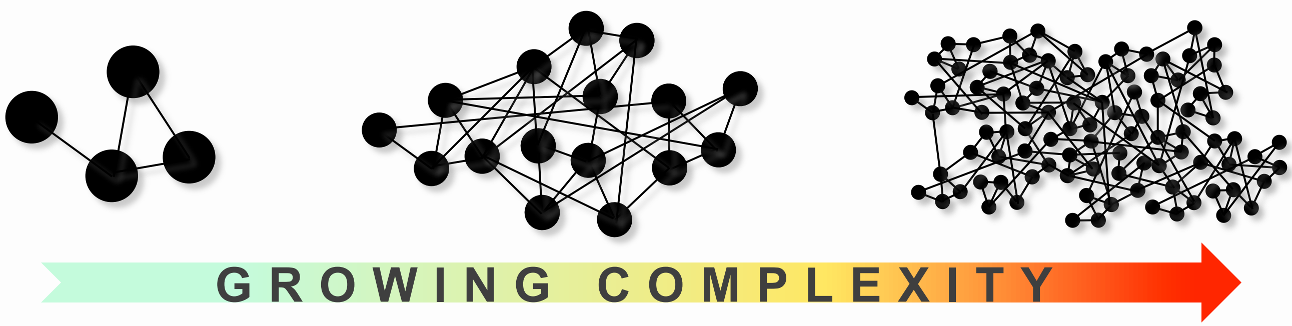


The Value in Thinking This Way

- If you start to think about the features you want as ***things that must emerge*** through interaction…
- Then you can't help *also* wondering about ***other things that might emerge***, besides the ones you intended
  - Whatever produces one will inevitably produce the other as a side effect
  - You must always worry about both
- *How would you know?!*

The Complexity Crisis

- As complexity grows, the number of potential interactions grows disproportionately



- Each layer removes us further from core analytical capabilities
- Confidence diminishes in explaining how things work *a priori*
- Even “correct” designs surprise us routinely

Complexity ⇒ Misunderstanding

- **Complexity** is basically a measure of how hard something is to **understand**
  - Variety, connectivity, depth, instability, opacity, intricacy, uncertainty, ambiguity…
  - Applies to **both analysis and communication**

- Complexity occupies the space between understanding and reality
  - For a complex system to succeed, many things have to be done right
  - However, a complex system can fail, even when all its parts work as designed

The Central Problem…

- In both science and engineering:

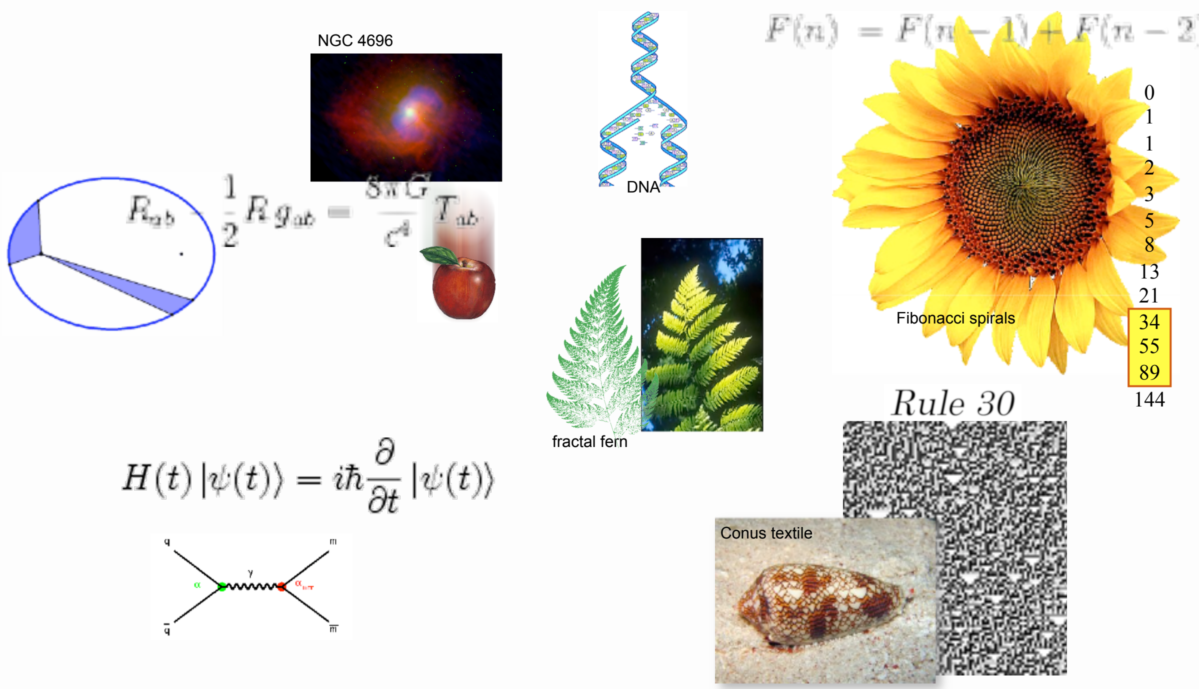
Find simple rules for complex behavior

- Rules are sought wherever there are **patterns**
- Patterns are expressions of the underlying rules
  - **Recurring structure**
    - Invariants among items, which may appear on the surface to be different
  - **Layered descriptions**
    - Ideas explained in terms of what's already understood
  - **Separation of concerns**
    - Limits on what must be considered at one moment

  etc.

Good Patterns…

- Not only describe — they explain!
  - As theories improve, they tend to become conceptually more abstract and layered
  - So the rules at each layer can become simpler



In Engineering
The Same Principles Apply

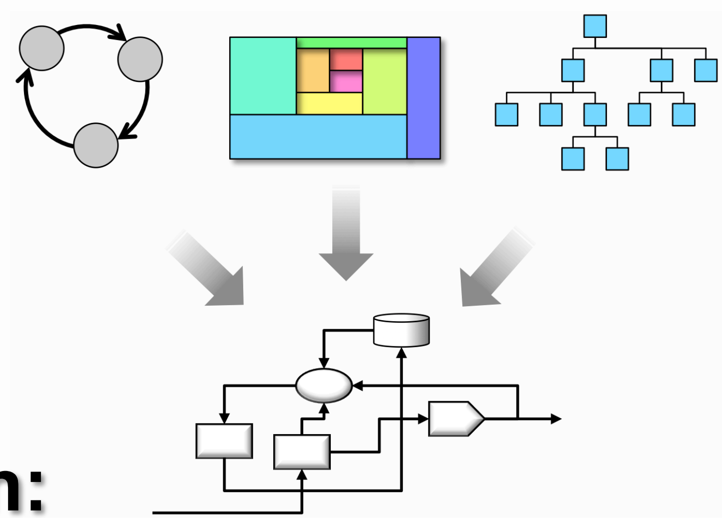
- Patterns impose order
  - Recurring Structure —
    - Mass production, standards for interface/form/process…
  - Layered Descriptions —
    - Hierarchical system design, protocol stacks…
  - Separation of Concerns —
    - Functional decomposition, weak coupling, modularity…
- Order fosters understandability

- These are the organizing **Concepts** of the architecture

Concepts Can Get Lost

- Each part of a system participates in many concepts
- This many-to-few mapping is responsible for troublesome entanglement of concepts in a complex system

Basic Concepts:



Complex Realization:

Example

An IMU is not merely a unit satisfying many disparate requirements flowed down “from above”

It is…

- a sensor in a control concept
- a region in a fault containment concept
- a load in a power concept
- a critical item in a safing concept
- a node in a networking concept
- a ward in a shielding concept
- a source in a telemetry concept,

and so on

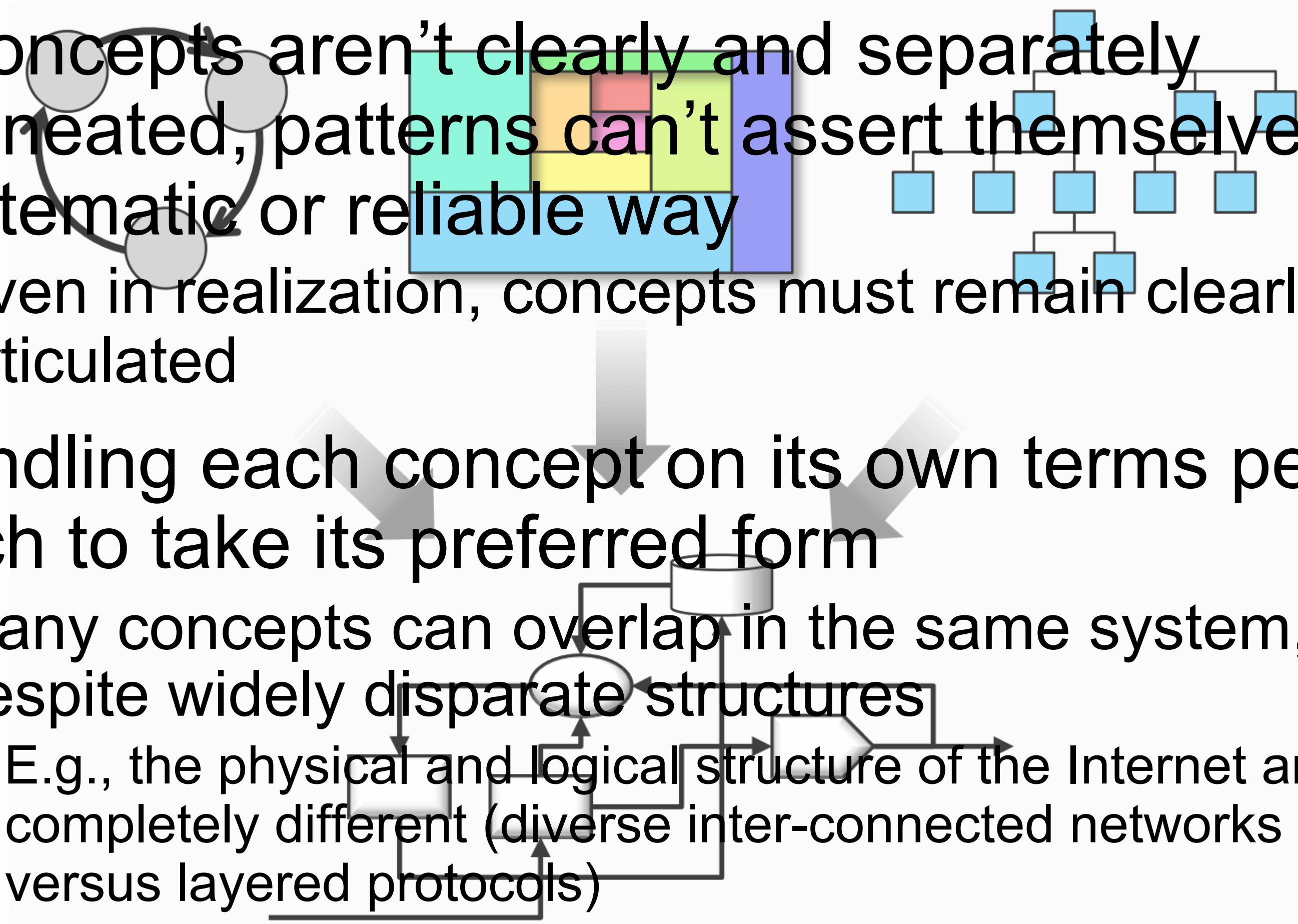
Many more **conceptual** parts than **realizational** parts

Nonetheless, Realization Seems To Rule!

- We tend to describe concepts in terms of their concrete implementations, rather than basic ideas
  - Levels gets flattened
  - Disparate concerns are swept together
  - Attention shifts from similarities to differences
  - General rules are replaced by point design descriptions
- **Complexity moves in to exploit inattention to pattern**

Concepts Need Space

- If concepts aren't clearly and separately delineated, patterns can't assert themselves in a systematic or reliable way
  - Even in realization, concepts must remain clearly articulated
- Handling each concept on its own terms permits each to take its preferred form
  - Many concepts can overlap in the same system, despite widely disparate structures
    - E.g., the physical and logical structure of the Internet are completely different (diverse inter-connected networks versus layered protocols)



Pattern versus Design

- Conceptual patterns *must* retain prominence throughout the lifecycle
- The **rules** that give rise to these patterns comprise a set of **constraints** on what we can design
  - They tell us both **what the design can and cannot be**
  - They allow as design only **what can be analyzed or validated**
- They help us see **what is essential** to a design concept
- It is from such **rules** and **exclusions** that **engineering elegance** is possible — without which…
  - Systems become increasingly muddled with incidental complexity
  - Piecemeal, ad hoc accommodations gradually ossify designs
  - Understanding becomes increasingly difficult
  - Shortfalls in functionality and efficiency are inevitable

However,
Not All Patterns are Created Equal

- We are awash in engineering “patterns”
  - Projects generate thousands of pages of design description in many forms
  - They describe modules, hierarchies, protocols, design requirements, processes, and so on — eventually in great detail
  - There are schemes for bus communications, power & grounding, fault containment, sequence coordination, time synchronization, and on and on
- It’s a mixed story
  - Some work a lot better than others
  - Some are arbitrary
  - And some old standbys are notoriously poor
- Many, however, have no clear conceptual delineation
  - We know something important is happening, but…
  - Like undiscovered Laws of Nature, they have no explanatory power

Lessons from Nature

- Complex, engineered systems are understandable only if well-chosen patterns are imposed to make understanding possible

- We seek patterns that are…
  - **Stable** — won't need frequent revision
  - **Fundamental** — broadly address important issues

- As in nature these tend to be **simple**
- But being **complete** and **consistent** are also essential

Also Important…

- Good patterns adhere strongly to aesthetics, experience, and fundamental **principles**
- Their rules enable **modeling** of adequate form & fidelity to address all attributes of concern
- They are easily explained, so that **compliance** can be required and verified

- In other words, we choose the patterns that permit us to demonstrate with confidence the correctness and suitability of our concepts

Good patterns
make such understanding practical

A Fault Management Example

Typical Fault Management Notions

"Concepts"

- Fault Tree, Failure Modes & Effects Analysis
- Error, Fault, Failure
- Threshold, Event, Persistence
- Detection, Monitor, Isolation, Response
- Priority, Level
- Critical Period, Mark & Rollback
- Safing

etc.

"Patterns"

- Monitors trigger responses
- Every monitor and response can be disabled
- Responses terminate command sequences

etc.

"Principles"

- Respond only to unacceptable conditions
- Avoid hair triggers and retriggering
- Tolerate false alarms
- Make parameters commandable
- Corroborate before severe responses
- Ensure commandability and long term safety
- Preserve consumables and critical data
- Log events and actions

etc.

Fundamental?

- Not Really
  - Imprecise and fragmented concepts
  - Weak patterns and principles
  - Exceptions and omissions
  - Cluttered with incidentals

  - Part of an even larger collection of interrelated notions in system management
  - Yet generally implemented separate from them

- No concise "Theory of Fault Protection"

A Sample Conceptual Mapping Issue

- Persistence threshold value:
  - Appears in monitoring functions, but is it…
  - Likelihood, transient duration, system error tolerance, response delay, false alarm avoidance, or what?

- Role depends on assumed meaning
  - Detection in state estimation
  - Branching in control decisions
  - Precedence among objectives
  - etc.

Back to Basics

What Does Fault Management Do?

- **Observes the system** *(measurements…)*
- **Uses models** *(failure modes…)*
- **Estimates system state** *(health, hazards…)*
- **Choses and coordinates actions** *(conflicts, resource use…)*
- **Directs the system** *(commands…)*
- **Meets system objectives** *(safety, viability, critical events…)*

Fault Management is *part* of an *integrated* Control System

Cognitive Control Fundamentals

Concepts

- Objectives on state
- Models of state behavior
- Knowledge of state
- Closed control loops on state

Patterns

- Each system state is assigned a cognizant control system
- Control systems interact via explicit state knowledge and coordinated objectives
- Knowledge and control designs exploit models

Principles

- Make objectives explicit, complete and clear
- Uniquely assign responsibility for all objectives on a state
- Make model usage apparent and consistent
- Explicitly coordinate concurrent objectives
- Keep state estimation independent of state control
- Represent state knowledge uncertainty openly and objectively
- Strive for a single source of truth for state knowledge
- Make control decisions based only on state knowledge and objectives

Differences in Perspective

When Concepts Retain Prominence

- "Fault management" detects and responds to faults
- **Fault tolerant control systems** achieve important system objectives, even when faults happen

- "Fault management" is verified by testing all monitors and responses
- **Fault tolerant control systems** are verified by showing how well they guard expectations of system performance

and so on

Resilience Architecture

- What are the patterns and principles of resilience?
  - If there is not theory for fault tolerance (or other matters), how could there be one for resilience?
- Is overall architectural integrity a prerequisite for resilience?
  - If an architecture can't easily be understood, how could one claim it is resilient?
- How can architectural concepts for resilience be integrated without losing their integrity?
  - If the patterns and principles of resilience aren't apparent in the system, how would one know they are still there?

Conclusion

Resilience starts with strong concepts

Resilience ends when conceptual integrity is lost

Practice principled architecture!