



Engineering Complex Ground Station Systems

A decorative graphic consisting of a horizontal line with a gradient from dark blue to yellow, ending in a large, stylized, orange and yellow arrowhead pointing to the right.

John Salasin
DARPA



Problem

- **Rumor -- Activation of Cooperative Engagement Capability (CEC) shuts down weapons control computers on all ships in battle group**

Software glitches leave Navy Smart Ship dead in the water (GCN July 13, 1998) "We are putting equipment in the engine room that we cannot maintain and, when it fails, results in a critical failure". It took two days of pier side maintenance to fix the problem.



Glitch in combat systems software knocks out weapons capability (The Virginian-Pilot, July 8, 1998) The cruisers Vicksburg and Hue City remain able to go to sea but may be unfit for battle for up to two years, while engineers try to mesh the hodgepodge of computer programs needed to run their combat systems.

Today: Cost and difficulty of change proportional to size of system



Importance

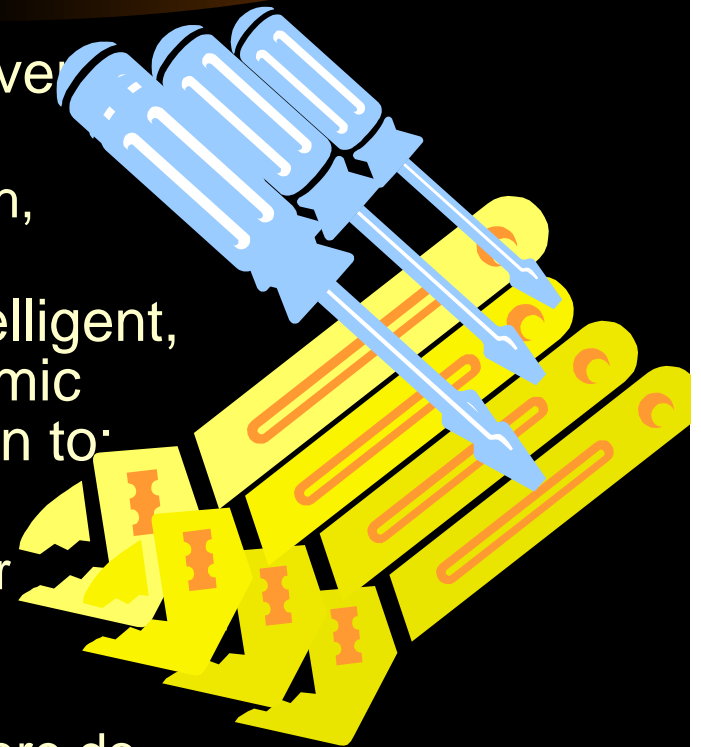
- Systems (of systems) are getting more difficult to understand, build, operate, and evolve
 - tighter integration
 - higher performance
 - interwoven concerns about reliability, safety, security, etc.
 - increased use of “black box” objects (e.g., COTS)
- We have fewer trained people who can understand, build, operate, and evolve them



Autonomous, Dynamic Systems Pose Problems (We can't specify everything)

Consider a nut, bolt, and tightening operation:

- For mechanical system (nut, bolt, screwdriver, wrench) we specify:
 - diameter, blade or Phillips head, screw pitch, torque
- How do things change if we employ an intelligent, autonomous electric screwdriver in a dynamic system? We also may need to pay attention to:
 - who initiates operation?
 - who determines if operation is successful or unsuccessful?
 - how is operation terminated?
 - what are the status or error messages? where do they go?
 - what resources are used, how are they monitored?
 - is there communication between nut and bolt, what protocol is used?





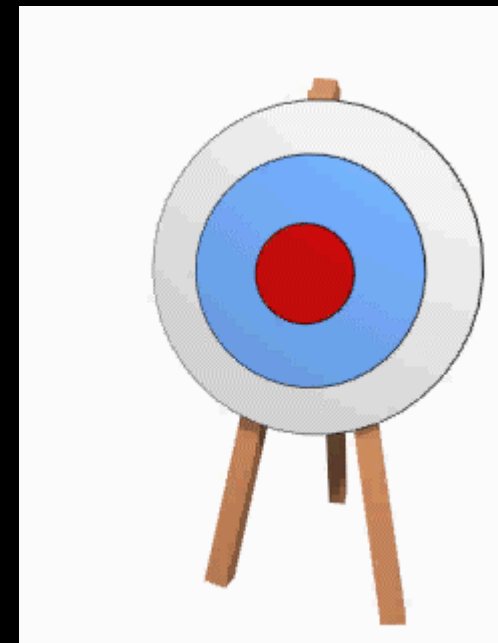
Why are systems complex?

- **Complexity results from:**
 - Only partial understanding of relationship between physical structure and behavior of the system. A large number of types of components and connections with weak central control.
 - Multiple, essential, interdependent behaviors (e.g. to support functional, safety, reliability, security requirements).
 - Dynamic behavior in both component behavior and typology; survivability, changing operating conditions require adaptation, self-correction.
- **We can't completely model them. Therefore, we can't:**
 - Understand them
 - Predict them
 - Control them
 - **Automatically compose or adapt them**



Humans cope with complexity by:

- **Using different models (or views) to focus attention on specific groupings of attributes / functionality, e.g.**
 - Timing behavior (or power consumption) in avionics system
 - Message throughput in packet switched network
- **Using multiple levels of abstraction to focus attention on different behavior**
 - Missile level (range, accuracy, pay load)
 - ATR system level (structural models of signal flows showing ATR configurations, resource models characterizing available hardware resources and allocation constraints).
- **Agreeing on standards**
- **Having recovery modes/strategies**





Vision -- Building on current capabilities

- **System engineering tools reduce apparent complexity:**
 - Multiple views, tailored to domain of concern, e.g., functionality, power, performance, maintainability, reliability, safety
 - Multiple levels of abstraction (with guarantees of correct mapping / refinement)
 - Express (and enforce) constraints
- **Tools and notations provide improved understanding of:**
 - How components, and models of components, interact (or interoperate)
 - Component integration standards, e.g.:
 - rights and responsibilities of each component
 - capabilities provided by shared infrastructure
- **Large savings due to:**
 - Early error detection
 - Automated generation and adaptation
 - Predictable composition (key to reuse)



Vision -- New Capabilities

- To drastically reduce cycle time, we need capabilities to:
 - compose systems at run-time from reusable components
 - dynamically assemble, reconfigure, and evolve systems
 - easily introduce new components to an existing design to add new functionality
 - adaptively and dynamically scale systems up and down in size to match problem size
 - get solutions off the shelf, combining components from multiple vendors
 - continuously upgrade components to assure best-of-class components

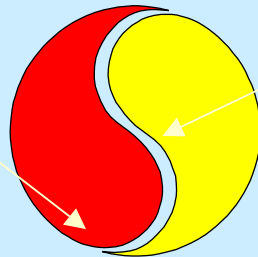


Next Systems Engineering Paradigm: Dynamic Interoperability

System = Functional + Non-Functional Properties

Functionality

- scenarios
- ontologies
- methods
- user case analysis
- user interfaces



Predictability of behavior

- optimized performance
- non-functional attributes (reliability, availability, etc.)
- quality of service (throughput, response time, etc.)
- temporal correctness (live-lock & deadlock freedom, etc.)

Today – Methodology

- Requirements analysis
- Specification
- Design
- Implementation
- Verification/Testing
- Deployment
- Re-engineering

Tomorrow – Technology

- Semantic advertising of component functionality
- Definition of constraints and behaviors
- Dynamic identification of components
- Run-time composition
- Monitoring and adaptation
 - reconfiguration (response to events)
 - evolution (response to requirement changes)



Some Critical Technologies

- Architecture / Composition
 - Synthesis
 - Analysis
- Model-based systems
 - Assure design refinement
 - Manage run-time adaptation



Architecture?

- **Architecture Notations (to model components, connectors, typology, behavior, constraints) -- Being extended to:**
 - support run-time dynamism (modification)
 - model dynamic systems
 - Simulator and Animator
 - Test Harness Driver (automating much of testing)
 - Optimizer
 - monitor constraint satisfaction (and conformance):
 - during design refinement
 - during operation
- **Scalable demonstrations of architecture -based tools (ensuring component interoperability in dynamic systems)**



Why is architecture useful?

- Enables automatic analysis and early detection of errors (correctness)
- Enables reuse and product line development
- Supports incrementality
- Supports optimization (non-functional attributes)



Why is architecture useful?

- **Enables automatic analysis and early detection of errors, e.g.:**
 - Given sequence of events can (or cannot) occur
 - Deadlock, livelock conditions
 - Sequence of processing steps in distributed applications (e.g., authorization completed before data is accessed)
 - Components can (or cannot) be composed with predictable properties, e.g.:
 - Timing
 - Resource use, starvation
 - Control of dynamic interaction (reconfiguration behavior) to ensure, e.g.:
 - Components exist before invocation
 - Links don't exist to non-existent components
 - Results / data not lost



Why is architecture useful?

- Enables reuse and product line development
 - Provides a transferable, reusable abstraction of a system and unambiguous specification of architectural standards (e.g., for HLA, ATIS, ...)
 - Provides infrastructure for very high level domain-specific notations / languages (and generating code)
 - “Style” provides vocabulary (component and connector “types”) and grammar (rules for legal interactions)
 - “General purpose” Architecture Description Languages (ADLs) allow disciplined design of systems that mix multiple styles
 - Support to defining families of systems -- foundation of software developed as a product line
- Provides assurance that implementation is a valid instantiation of architecture



Why is architecture useful?

- Supports Incrementality
 - Dynamic (run-time) modification
 - Specification and control of change mechanisms
 - Assurances that properties can be relied upon while the system evolves
 - Assuring properties at architecture, rather than code, level
 - Automated code development / evolution
 - Architecture modification => code modification
 - Automated support to test and analysis
 - Basis for specifying / deriving test and analysis plans (modifications)

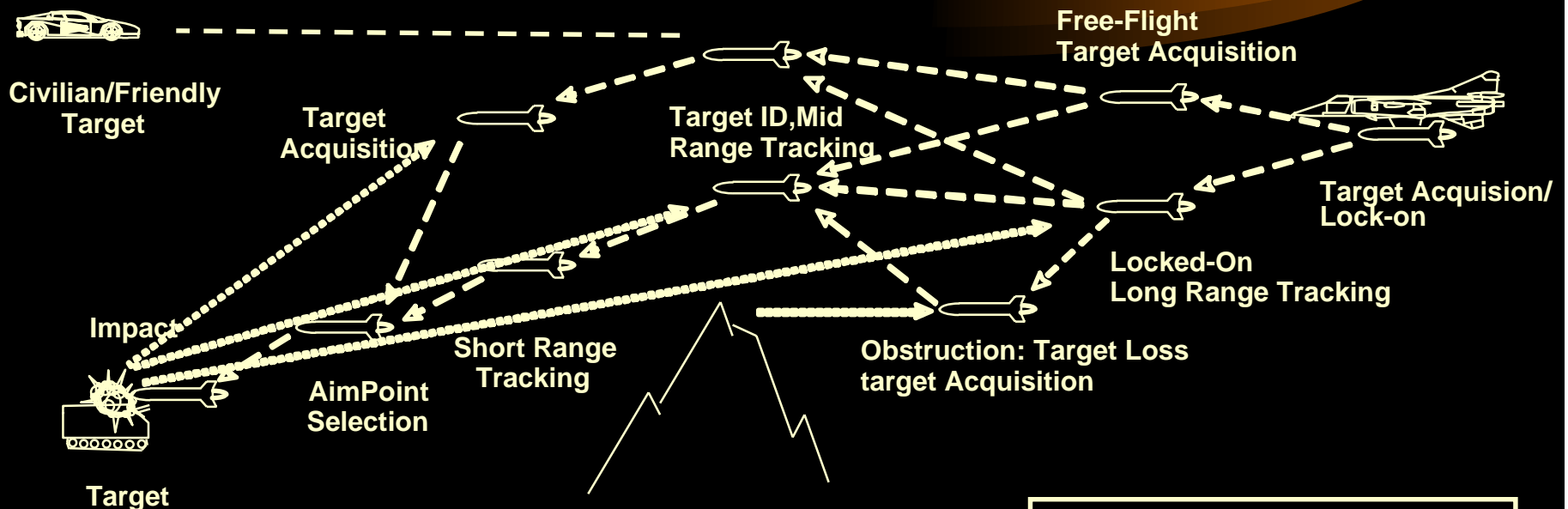


Why is architecture useful?

- **Supports optimization of component interaction wrt, e.g.:**
 - Performance
 - Fault tolerance
 - Security/safety concerns



Why models are important



Complicating Factors:

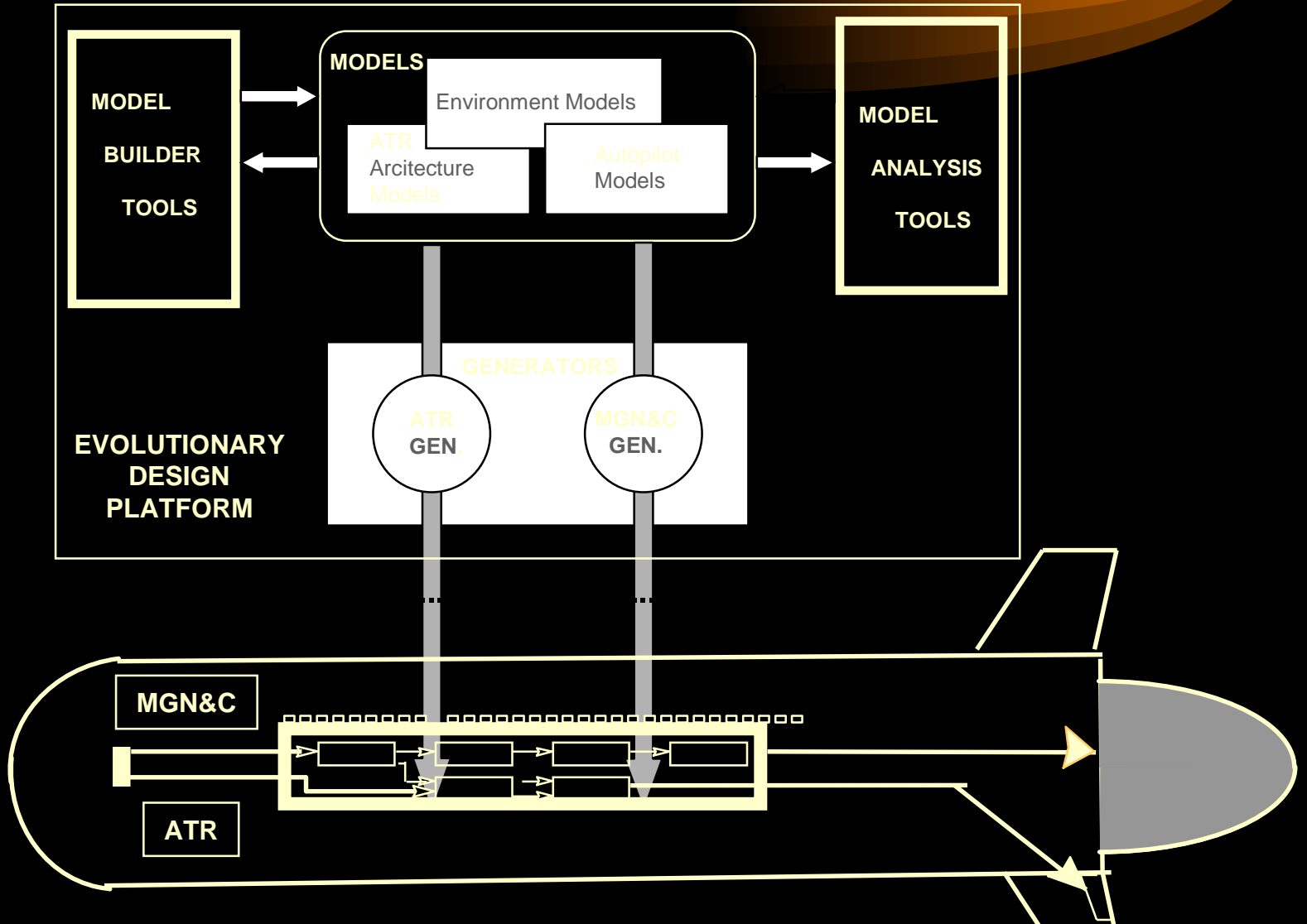
- Hardware Size/Weight/Power Limited by Platform (Pre-Launch(Initial Acquisition/lock-on) is very power-limited)
- Obstructions & Environmental factors cause major mode shifts
- Algorithms & Data Rates Drastically Change Between Modes (Different Architectures Needed for Best Performance)
- Long Design Cycles (Obsolete Components Used)

---> Highly Adaptive System With Evolvable Design needed.

Missile Targeting
Mode Evolution



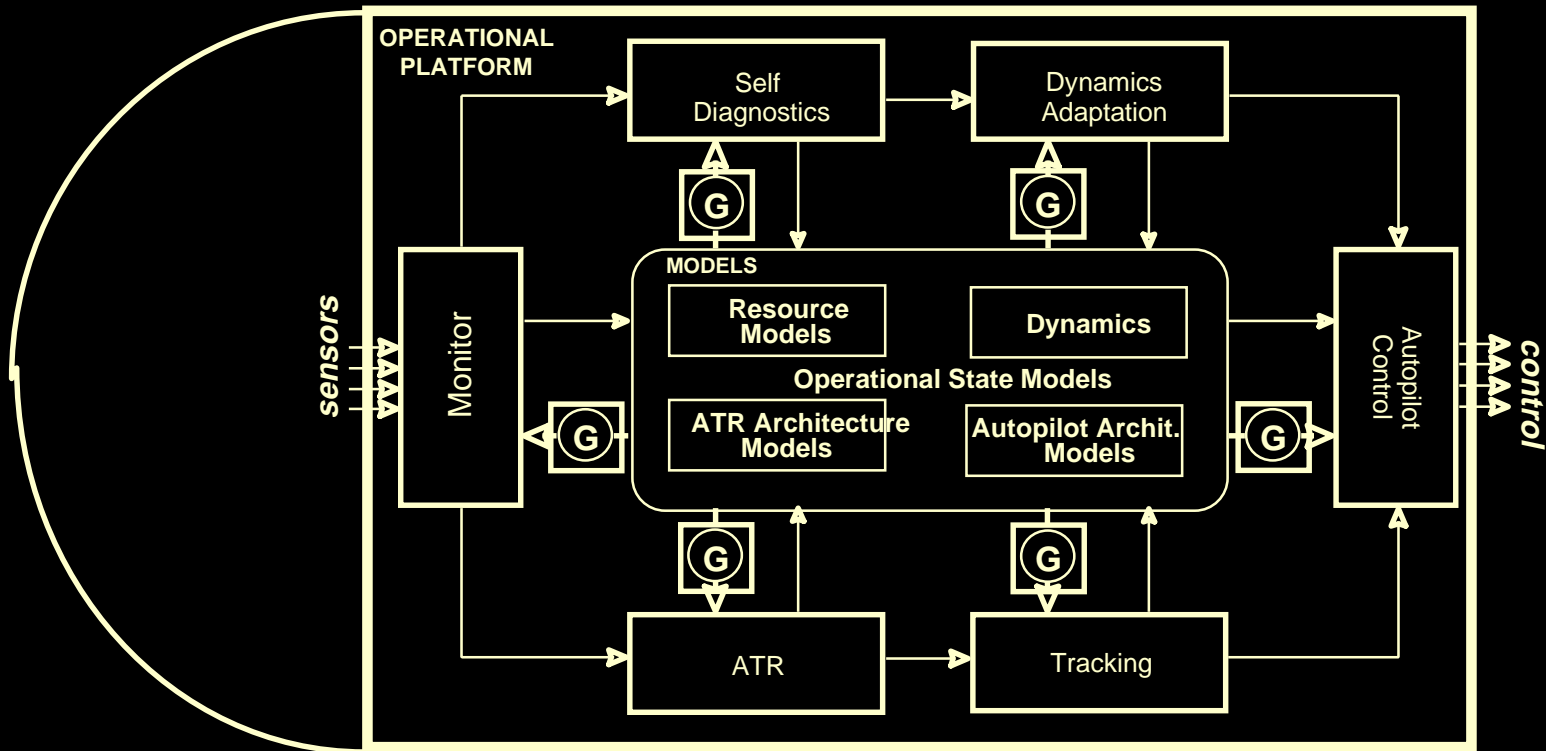
Models in missile GN&C today





Models in missile GN&C tomorrow

Put the model in the missile!





Multiple Models are Needed

- To represent:
 - Functionality
 - Resource use
 - Other attributes related to QoS (e.g, safety, fault tolerance, reliability, survivability).
 - Evolution capability / restrictions
 - Implementation: How can an implementation be obtained from the architectural model?
 - Standards / Consistency : Does the system conform to externally specified standards?
Does the system obey a given set of rules?



Model-based Themes

- Integration of evolving design models (with multiple views) with system management functions to provide adaptive management and self-correction.
 - Reduce complexity without losing track of details.
 - Improve ability to select and use the best models, improve models, and to evolve models
 - Make the cost of adopting this technology feasible.



Some Needed Modeling Technology

- Dynamic models of dynamic systems with varying views and levels of fidelity (and languages/tools to create them and reason about them).
- Tools to support integration of models and system management -- providing self-modification based on model manipulation
- Decision rules for model modification (including safety or other constraints)
- Ability to reason about systems (or architectures) in context (e.g., hardware environment)
- Automatic generation of connectors to facilitate model/system interactions.
- Derivation of models from existing systems (learning)



The challenge

To integrate left-brain and right-brain approaches

LEFT (Analytic)

Verbal

Responds to word meaning

Sequential

Processes information linearly

Responds to logic

Plans ahead

Predictability of behavior

- optimized performance
- non-functional attributes
(reliability, availability, etc.)
- quality of service
(throughput, response time, etc.)
- temporal correctness
(live-lock & deadlock freedom, etc.)

RIGHT (Global)

Visual, tactual, kinesthetic

Responds to pitch, feeling

Random

Processes information in chunks

Responds to emotion

Spontaneous

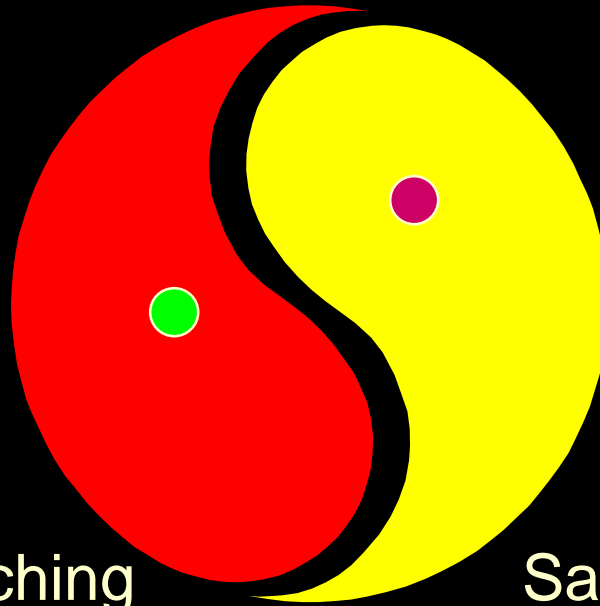
Functionality

- scenarios
- ontologies
- methods
- user case analysis
- user interfaces



The challenge

Combine left-and right-brain, but emphasize different techniques for different requirements



Creative web searching
Plan development
Concept exploration

Safety critical systems
Machine control
Avionics