Systems Qualities Ontology, Tradespace, and Affordability (SQOTA)

Prof. Barry Boehm, USC: PI

SERC Sponsor Research Review
November 8, 2018

Outline

- ➡ SQOTA Foundations: Resilience and the SERC System Qualities (SQs) Ontology
  - SQs Ontology origins, structure
  - The ontology and the five types of resilience
- SQOTA Models, Methods, Processes, and Tools
  - AFIT, NPS, PSU swarms of drones models and tools
  - GTRI, Wayne State system qualities analysis tools
  - Software technical debt (TD) data analytics
  - Parallel agile processes and tools
- Next-generation cost estimation models and tools

SYSTEMS ENGINEERING Research Center

SQs Ontology origins

- Engineered Resilient Systems a US DoD priority area in 2011
- Most DoD ERS activity focused on physical systems
  - Field testing, supercomputer modeling, improved vehicle design and experimentation
- SERC tasked to address resilience, tradespace with other SQs for cyber-physical-human systems
  - Vehicles: Robustness, Maneuverability, Speed, Range, Capacity, Usability, Modifiability, Reliability, Availability, Affordability
  - C3I: also Interoperability, Understanding, Agility, Relevance, Speed
- Resilience found to have numerous definitions
  - Wikipedia 2012 proliferation of definitions
  - Weak standards: ISO/IEC 25010: Systems and Software Quality
  - INCOSE System Engineering Handbook
  - Resulting SERC Systems Qualities Ontology

SYSTEMS ENGINEERING Research Center

Proliferation of Definitions: Resilience

- **Wikipedia 2012 Resilience variants: Climate, Ecology, Energy Development, Engineering and Construction, Network, Organizational, Psychological, Soil**

- **Ecology and Society Organization Resilience variants: Original-ecological, Extended-ecological, Walker et al. list, Folke et al. list; Systemic-heuristic, Operational, Sociological, Ecological-economic, Social-ecological system, Metaphoric, Sustainabilty-related**

- **Variants in resilience outcomes**
  - **Returning to original state; Restoring or improving original state; Maintaining same relationships among state variables; Maintaining desired services; Maintaining an acceptable level of service; Retaining essentially the same function, structure, and feedbacks; Absorbing disturbances; Coping with disturbances; Self-organizing; Learning and adaptation; Creating lasting value**
  - **Source of serious cross-discipline collaboration problems**

Weak standards: ISO/IEC 25010: Systems and Software Quality

- Oversimplified one-size-fits all definitions
  - Reliability: the degree to which a system, product, or component performs specified functions under specified conditions for a specified period of time
  - OK if specifications are precise, but increasingly "specified conditions" are informal, sunny-day user stories.
    - Satisfying just these will pass "ISO/IEC Reliability," even if the system fails on rainy-day user stories
    - Surprisingly for a quality standard, it will pass "ISO/IEC Reliability," even if system fails on satisfying quality requirements
  - Need to reflect that different stakeholders rely on different capabilities (functions, performance, flexibility, etc.) at different times and in different environments
- Weak understanding of inter-SQ relationships
  - Security Synergies and Conflicts with other qualities

Current SERC SQs Ontology

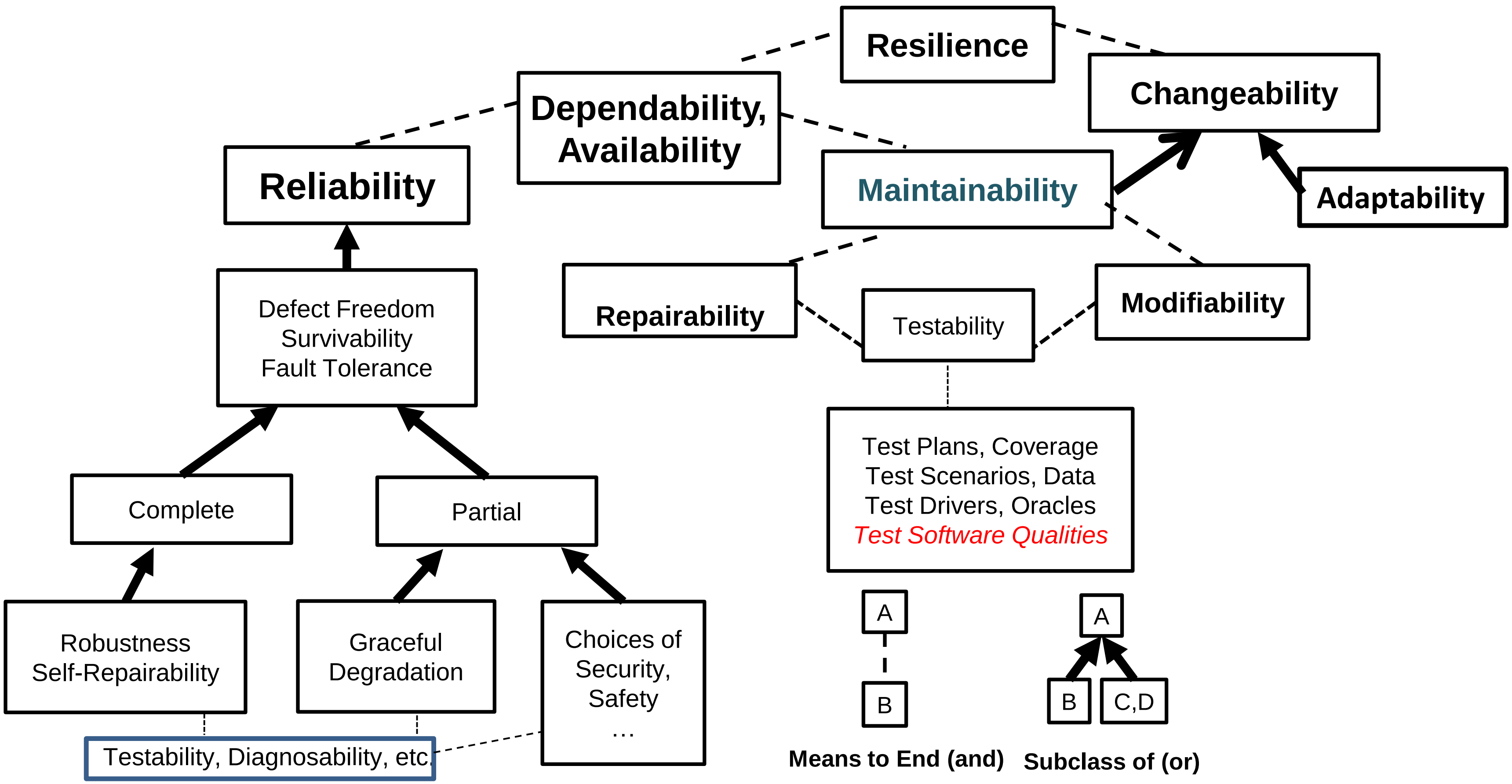
- Modified version of IDEF5 ontology framework
  - Classes, Subclasses, and Individuals
  - Referents, States, Processes, and Relations
- Top classes cover major stakeholder value propositions
  - Mission Effectiveness, Life Cycle Efficiency, Dependability, Changeability
- Subclasses identify means for achieving higher-class ends
  - Means-ends one-to-many for top classes
  - Ideally mutually exclusive and exhaustive, but some exceptions
  - Many-to-many for lower-level subclasses
- Referents, States, Processes, Relations cover SQ variation
  - Referents: Stakeholder-SQ value-variation (gas mileage vs. size, safety)
  - States: Internal (miles driven); External (off-road, bad weather)
  - Processes: Internal (cost vs. quality); External (haulage, wild driver)
  - Relations: Impact of other SQs (cost vs. weight vs. safety)

SYSTEMS ENGINEERING Research Center

Stakeholder value-based, means-ends hierarchy
Note key roles of Maintainability

- Mission operators and managers want improved Mission Effectiveness
  - Involves Physical Capability, Cyber Capability, Human Usability, Speed, Accuracy, Impact, Endurability, Maneuverability, Scalability, Versatility, Interoperability
- Mission investors and system owners want Life Cycle Efficiency
  - Involves Cost, Duration, Personnel, Scarce Quantities (capacity, weight, energy, …); Manufacturability, Maintainability
- All want system Dependability: cost-effective defect-freedom, availability, and safety and security for the communities that they serve
  - Involves Reliability, Availability, Maintainability, Survivability, Safety, Security, Robustness
- In an increasingly dynamic world, all want system Changeability: to be rapidly and cost-effectively changeable
  - Involves Maintainability (Modifiability, Repairability), Adaptability
  - Similar MIT Changeability ontology framework being coordinated
- Resilience a combination of Dependability and Changeability

Dependability, Changeability, and Resilience



How does Resilience depend on Maintainability?

Resilience: INCOSE SysE Handbook

- **Resilience is the ability to prepare and plan for, absorb or mitigate, recover from, or more successfully adapt to actual or potential adverse events.**
  - **Absorb: Robustness (e.g., via armor or redundancy)**
  - **Mitigate: Graceful Degradation**
  - **Recover from: Repairability**
  - **Adapt to actual or potential adverse events:**
    - **Internally: Self-modifiability**
    - **Externally: User-modifiability**

- **Activities in black are performed during Development. Subsequent upgrades are counted as Maintenance activity along with the activities in red.**

Some Surprise-Free Software Trends
with Resilience Implications

- More, larger, more complex software and systems
  – Internets of things, self-driving cars, …
- Increasing speed of change
- Increasing need for software dependability
  – Safety of cyber-physical-human systems
- Increasing software autonomy
  – Principle of Human Primacy in microseconds?
- Increasing data capture, data analytics
- Increasing legacy software, evolution challenges
  – Mounting technical debt

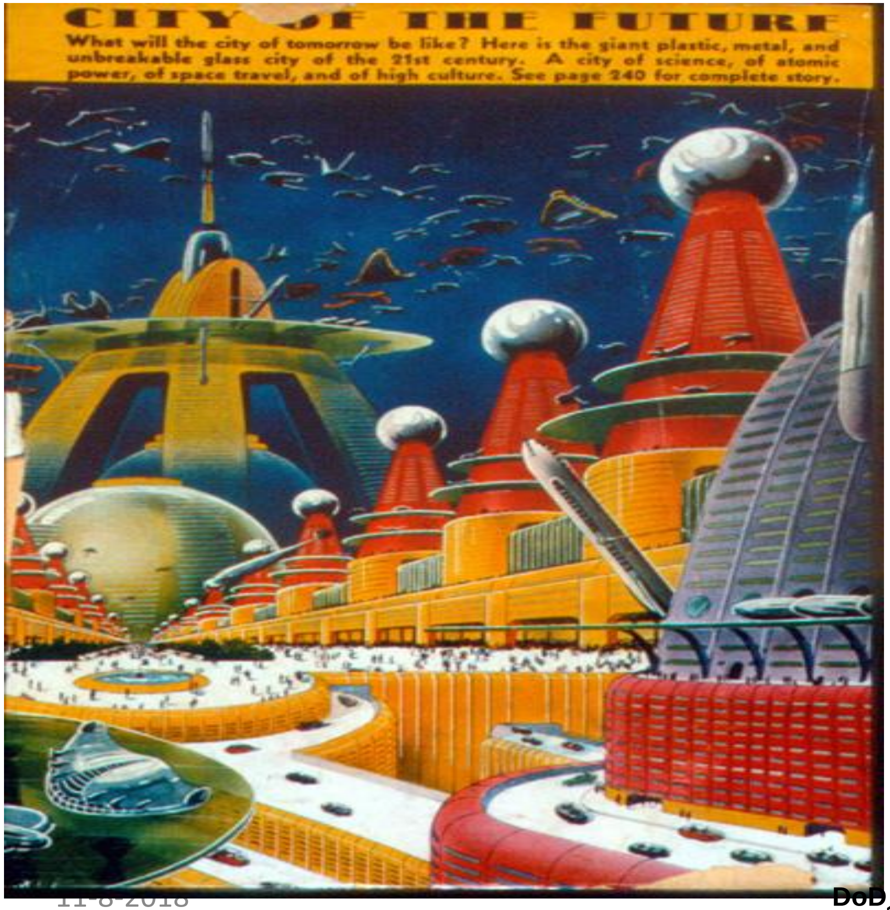
What is Technical Debt (TD)?

- TD: Delayed technical work or rework that is incurred when short-cuts are taken or short-term needs are addressed first
  - The later you pay for it, the more it costs (interest on debt)
- Global Information Technology Technical Debt [Gartner 2010]
  - 2010: Over $500 Billion; By 2015: Over $1 Trillion
- TD as Investment
  - Competing for first-to-market
  - Risk assessment: Build-upon prototype of key elements
  - Rapid fielding of defenses from terrorist threats
- TD as Lack of Foresight
  - Overfocus on Development vs. Life Cycle
  - Skimping on Systems Engineering
  - Hyper-Agile Development: Easiest-First increments
    - Neglecting Rainy-Day Use Cases, Non-Functional Requirements

Persistence of Legacy Systems

- New resilient technology needs to address improvement of aging legacy systems

1939's Science Fiction World of 2000



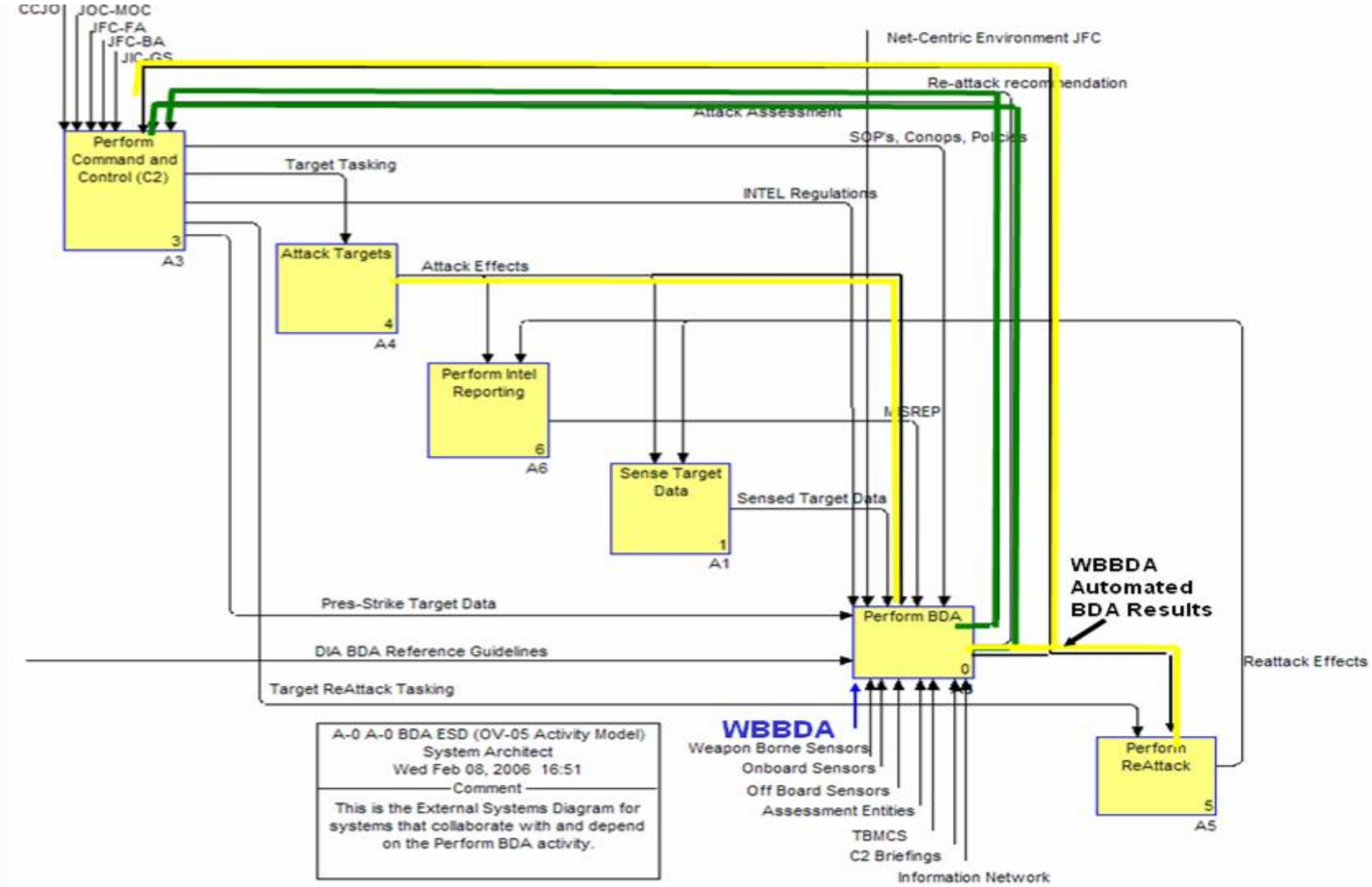
Actual World of 2000

SYSTEMS ENGINEERING
Research Center

Outline

- SQOTA Foundations: Resilience and the SERC System Qualities (SQs) Ontology
  - SQs Ontology origins, structure
  - The ontology and the five types of resilience
- ➡ SQOTA Models, Methods, Processes, and Tools
  - AFIT, NPS,PSU swarms of drones models and tools
  - GTRI, Wayne State system qualities analysis tools
  - Software technical debt (TD) data analytics
  - Parallel agile processes and tools
- Next-generation cost estimation models and tools

AFIT, NPS Swarms of Drones MODELS-I



Example UAS Missions

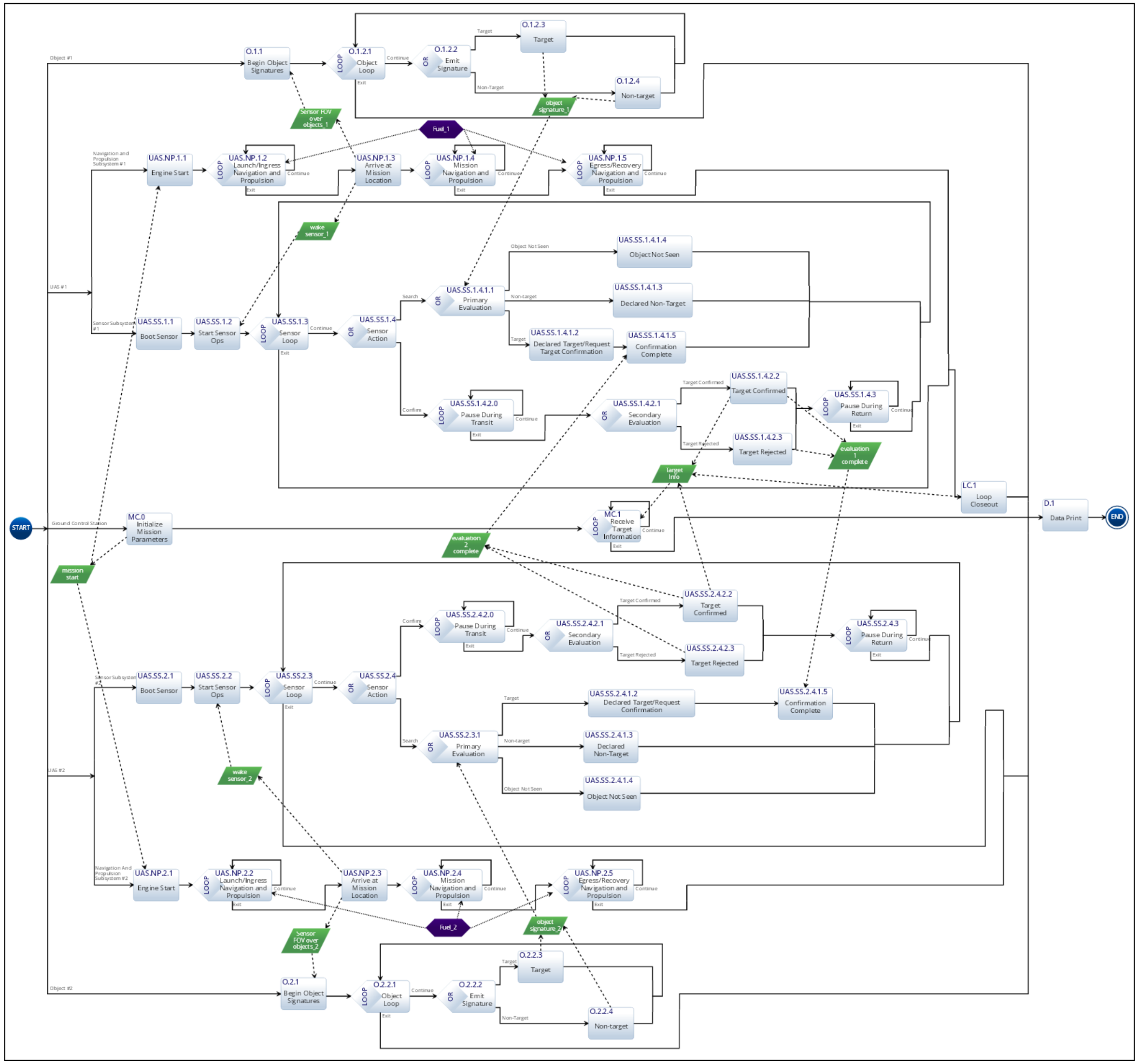
- Single UAS Search and Target Tracking (Simple Missio
- UAS Pair Search and Target Tracking
- Find, Fix and Finish Terrorist Leadership (1
- Find, Fix and Finish Terrorist Leadership (2)
- Mobile Missile Launcher Monitoring (1)
- Mobile Missile Launcher Monitoring (2)

Single UAS Simple Mission Threads

- **Launch**
- **Navigation and flight**
- **Search and target ID including evaluation**
- **Target tracking**
- **Return/recovery**

SYSTEMS ENGINEERING
Research Center

Example Activity Model (OV-5b) for Two UAS Mission



Resilience Contract (Madni, 2017*)

Flexible formal modeling construct with learning capability

for stochastic/probabilistic systems

partial observability, noisy sensors, uncertain environment

key trade-off: formality (V&V) vs flexibility (resilience)

developed at design time, trained during operation ("learning")

Resilience Contract (RC) comprises:

Traditional Contract (TC)

Partially Observable Markov Decision Process (uncertainty)

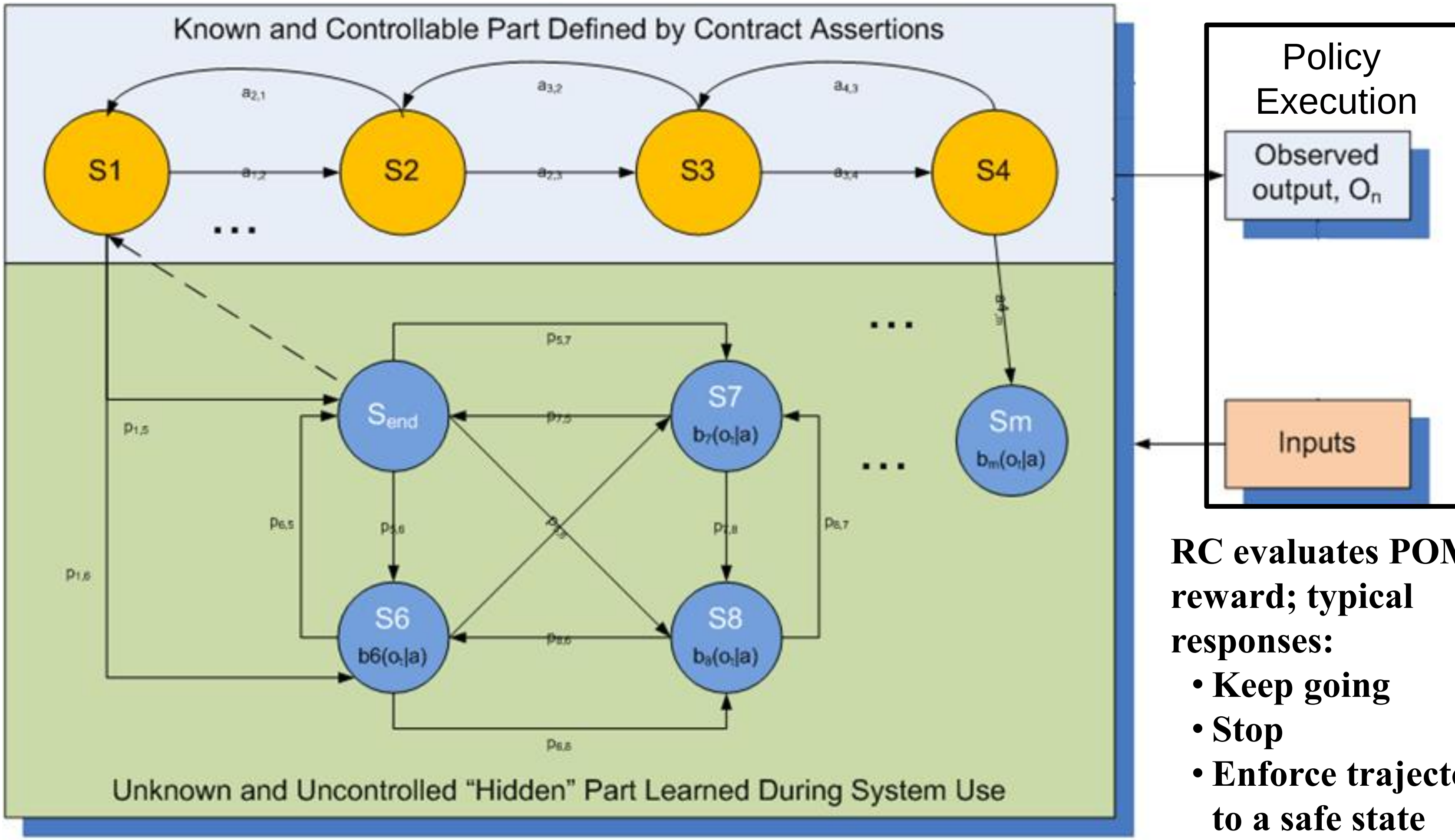
Reinforcement Learning (RL)

RC = TC + flexible assertions + POMDP + RL

in-use learning (hidden states, transitions, etc.) + pattern recognition

* Madni, A.M., *Autonomous System-f-Systems (Chapter10)* in *Transdisciplinary Systems Engineering: Exploiting Convergence in a Hyper-Connected World*, Springer, 2017.

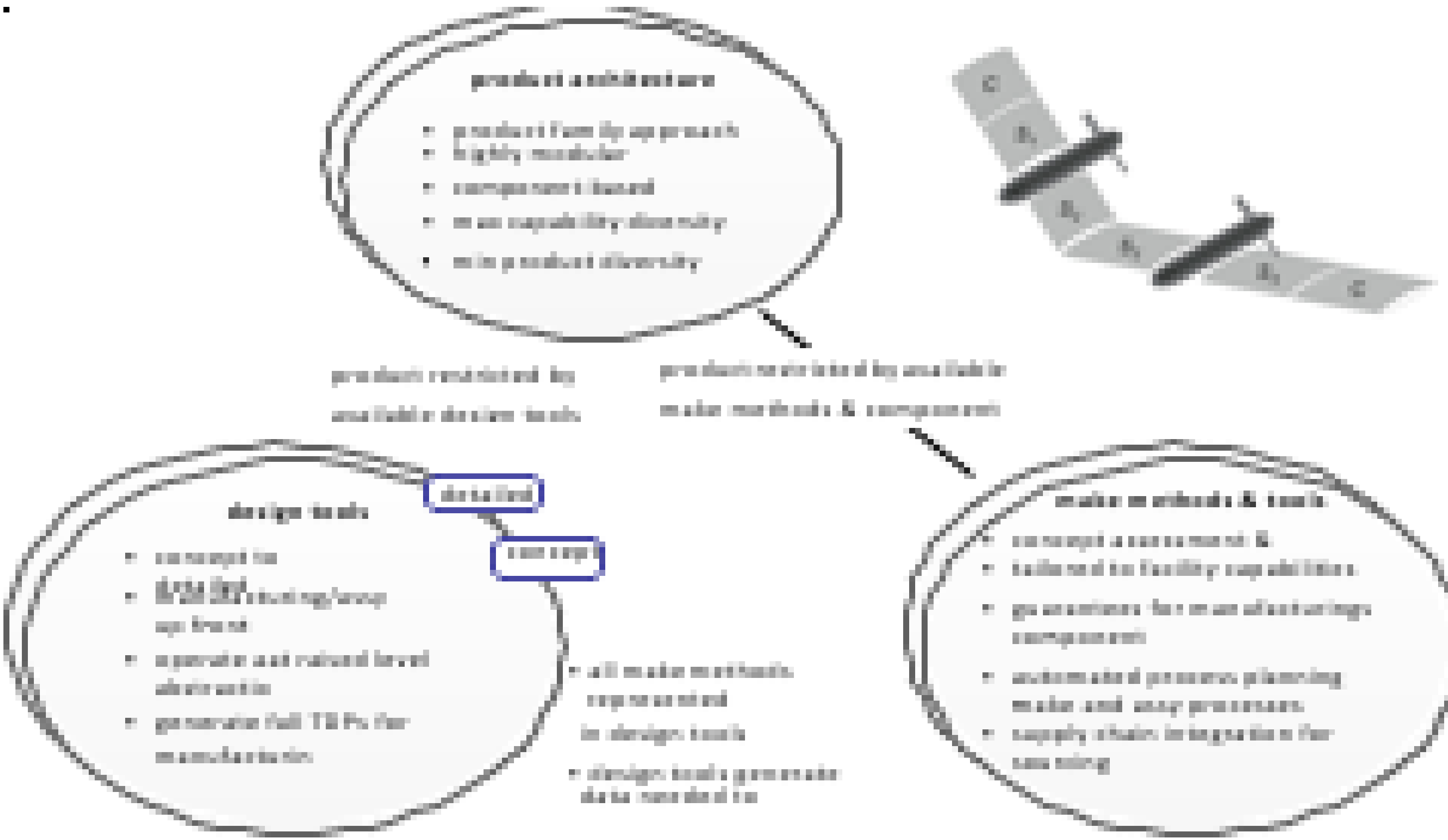
Resilience Contract (RC): Operational Concept



RC evaluates POMDP reward; typical responses:

- **Keep going**
- **Stop**
- **Enforce trajectory to a safe state**
- **Notify support team**

Penn State Drone Modeling



SYSTEMS ENGINEERING
Research Center

GTRI Tool Development, Transition - I

Scenario Based Needs Context

Competing Objectives in Parallel

Stakeholder 1: Prioritizes capabilities {A, B, C}

Stakeholder 2: Prioritizes capabilities {C, D, E}

Competing Objectives in Series

Prioritize capabilities {A, B, C}
At levels $\{A_{R1}, B_{R1}, C_{R1}\}$

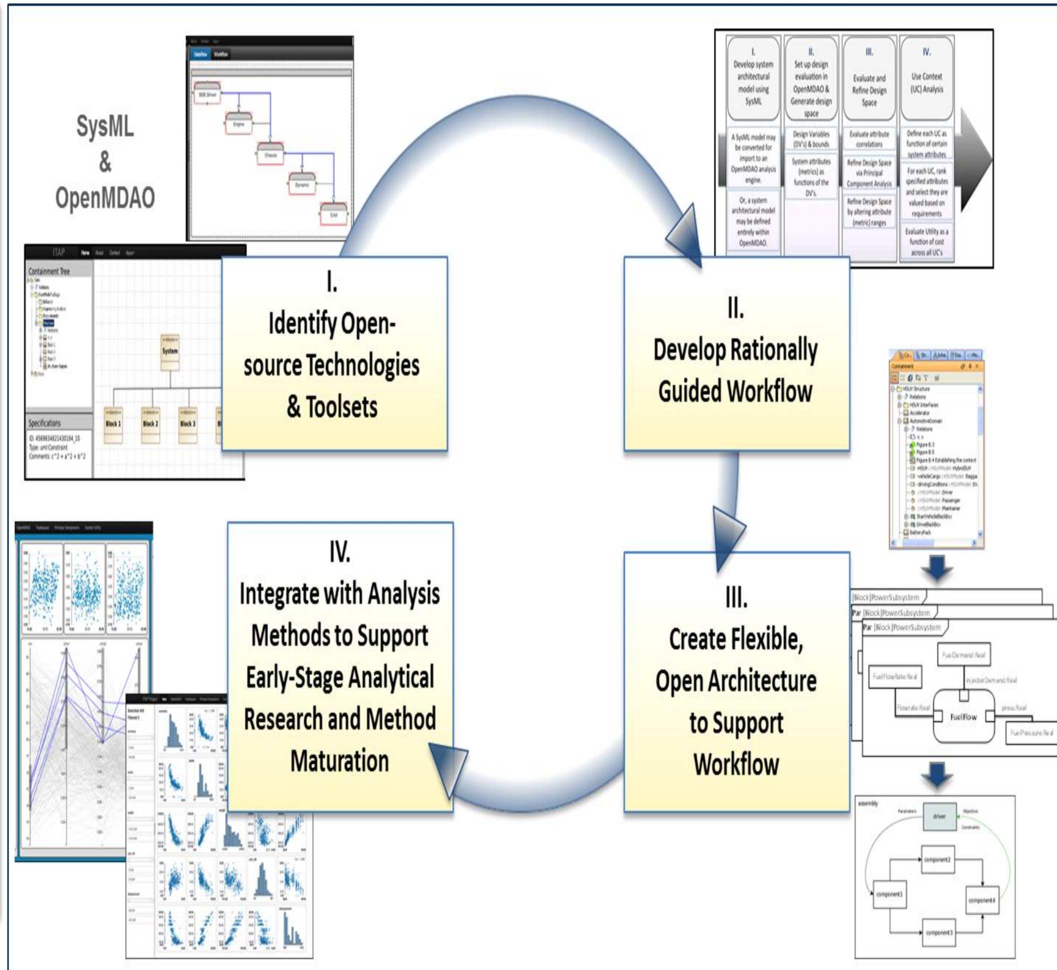
Prioritize capabilities {C, D, E}
At levels $\{C_{R2}, D_{R2}, E_{R2}\}$

System Service Life

Requirements or Mission Profile at Time 1

Requirements or Mission Profile at Time 2

Flexible workflows, integrating MBSE and MDAO



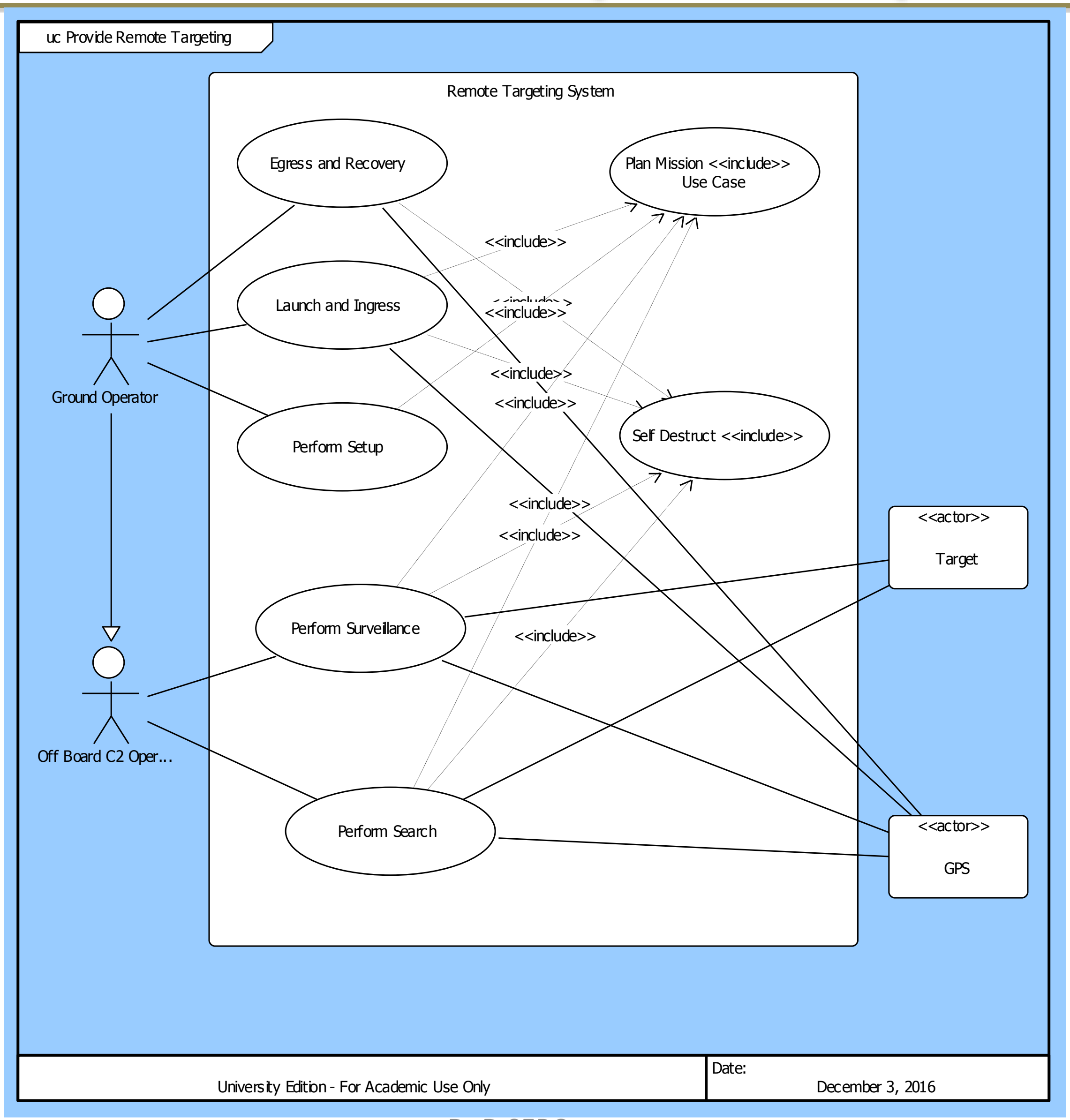
GTRI Tool Development, Transition - II

SYSTEMS ENGINEERING
Research Center

WSU Remote Targeting System (RTS) Background

- **Comparing two architecture variants of Remote Targeting System (RTS) performed by semi-autonomous vehicles**
- **Baseline variant can have multiple vehicles, but uses human-in-the-loop to declare targets**
  - **Requires data link back to operator for each vehicle**
- **Autonomous Target Recognition (ATR) variant has heterogeneous sensors, and can use multiple vehicles to auto confirm target declarations without requiring a human.**
  - **Vehicle needs an autonomous target recognition algorithm (ATR)**
  - **Vehicle requires a data link between vehicles, in addition to the data link back to the human operator (which must be modified to accommodate the ATR declarations);**
  - **The Plan Mission Use Case must be modified to include loading of target templates**
  - **Search Use Case must be modified to accommodate the ATR activities**
  - **Additional <<include>> Use Cases, “Perform ATR” and “Confirm Target” must be added**
  - **New and modified requirements must be accommodated**

Remote Targeting System (RTS) Scenarios (Use Cases)



SYSTEMS ENGINEERING
Research Center

Outline

- **SQOTA Foundations: Resilience and the SERC System Qualities (SQs) Ontology**
  - SQs Ontology origins, structure
  - The ontology and the five types of resilience
- **SQOTA Models, Methods, Processes, and Tools**
  - AFIT, NPS swarms of drones models and tools
  - GTRI, Wayne State system qualities analysis tools
  - ➡ Software technical debt (TD) data analytics
  - Parallel agile processes and tools
- **Next-generation cost estimation models and tools**

Software Quality Understanding by Analysis of Abundant Data (SQUAAD)

- An automated cloud-based infrastructure to
  - Retrieve a subject system's information from various sources (e.g., commit history and issue repository).
  - Distribute hundreds of distinct revisions on multiple cloud instances, compile each revision, and run static/dynamic programming analysis techniques on it.
  - Collect and interpret the artifacts generated by programming analysis techniques to extract quality attributes or calculate change.
- A set of statistical analysis techniques tailored for understanding software quality evolution.
  - Simple statistics, such as frequency of code smell introduction or correlation between two quality attributes.
  - Machine learning techniques, such as clustering developers based on their impact.
- An extensible web interface to illustrate software evolution.

SQUAAD Is Designed To Help...

- **Organizations**, determine
  - Which divisions and project types have better or worse quality.
  - Which quality attributes are being achieved poorly or well.
  - How do these correlate with customer satisfaction and total cost of ownership.
- **Managers**, better understand
  - Which types of projects or personnel contribute most to quality problems or excellence.
  - Which types of project events correlate with which types of quality increase or decrease.
  - Rates at which project is increasing or decreasing technical debt
  - Sources of technical debt and frequency of occurrence
- **Developers**, better understand and continuously monitor
  - Sources of technical debt and frequency of occurrence
  - Results of efforts to reduce technical debt

A Recent Experiment

Metrics

Group	Abbr.	Tool	Description
Basic	LC	SonarQube	Physical Lines excl. Whitespaces/Comments
	FN	SonarQube	Functions
	CS	FindBugs	Classes
Code Quality	CX	SonarQube	Complexity (Number of Paths)
	SM	SonarQube	Code Smells
	PD	PMD	Empty Code, Naming, Braces, Import Statements, Coupling, Unused Code, Unnecessary, Design, Optimization, String and StringBuffer, Code Size
Security	VL	SonarQube	Vulnerabilities
	SG	PMD	Security Guidelines
	FG	FindBugs	Malicious Code, Security

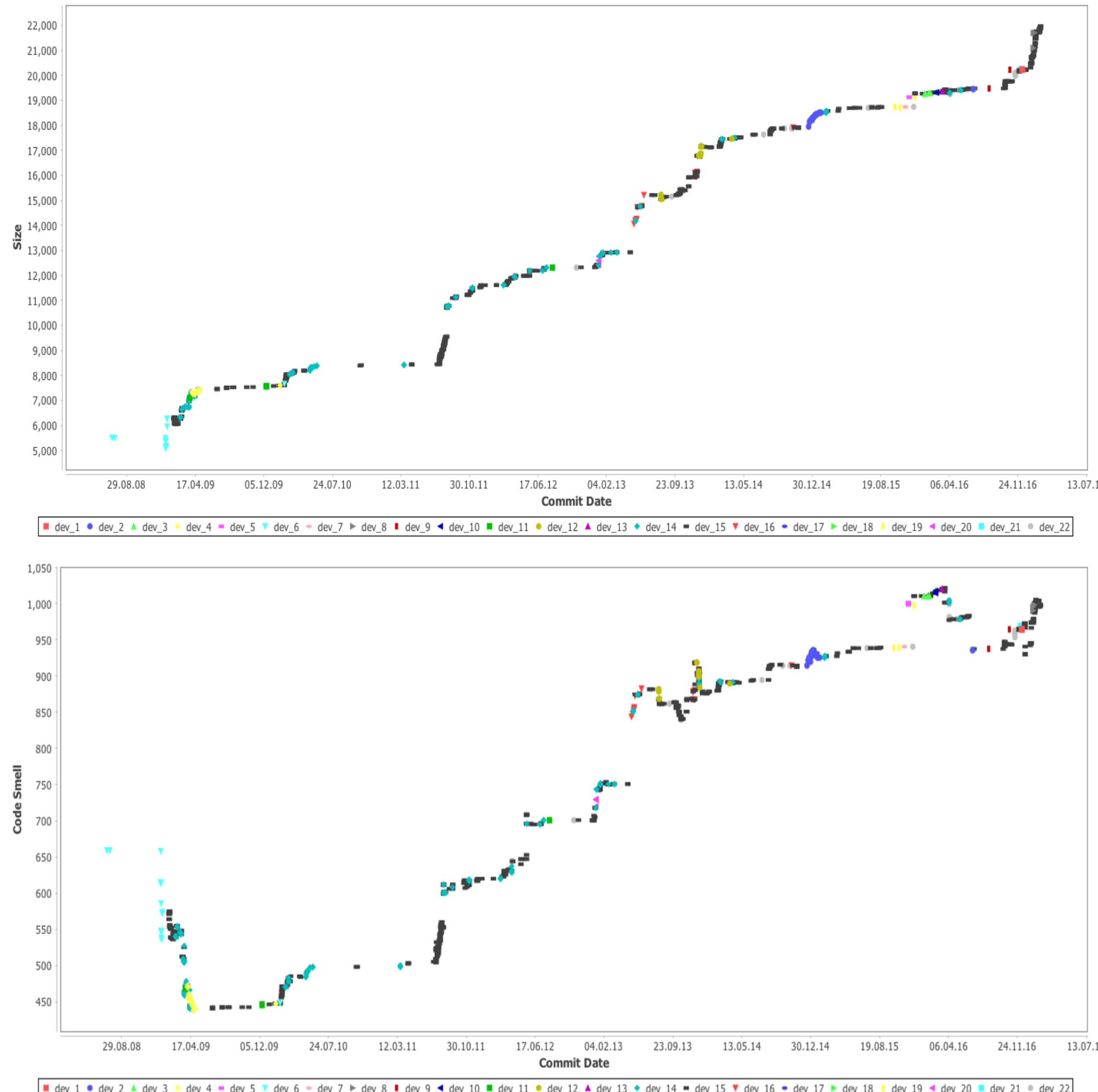
Scale

Org.	Time Span	Sys.	Dev.	Rev.	MSLOC
Netflix	09/12-12/17	12	251	3683	34
Apache	01/02-03/17	39	1102	20197	576
Google	08/08-01/18	17	402	11354	753
Total	01/02-01/18	68	1755	35234	1363

SYSTEMS ENGINEERING
Research Center

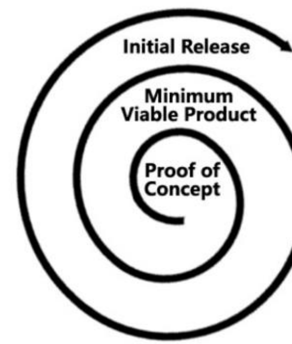
Evolution of a Single Quality Attribute

- How a single quality attribute evolves.
- Two metrics
  - Size (top)
  - Code Smells (bottom)
- One project
- 9 years

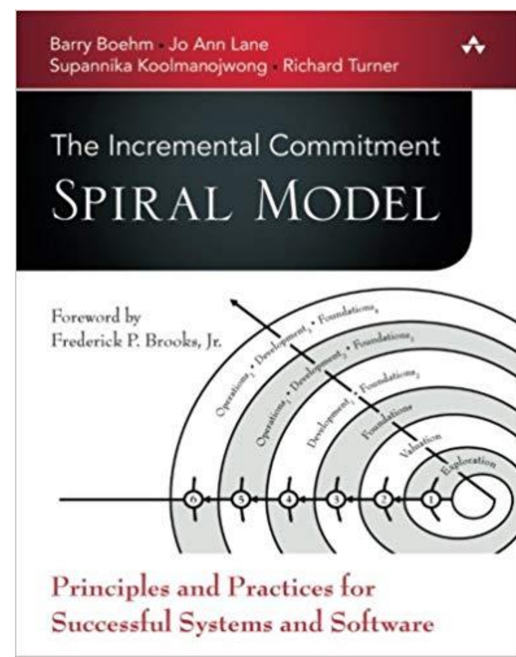


PARALLEL AGILE

Get to market faster without sacrificing quality



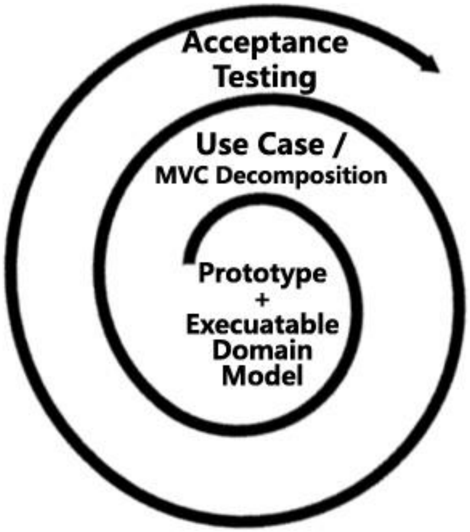
- 3 phases: Proof of concept, MVP, Initial Release
  - Each phase approximately a month long
  - Proof of concept uses prototyping to discover requirements, reduce risk
  - MVP uses UML modeling, details sunny/rainy day scenarios, reduce technical debt
  - Initial Release focuses on acceptance testing, performance tuning, optimization, reduce hotfixes



PARALLEL AGILE

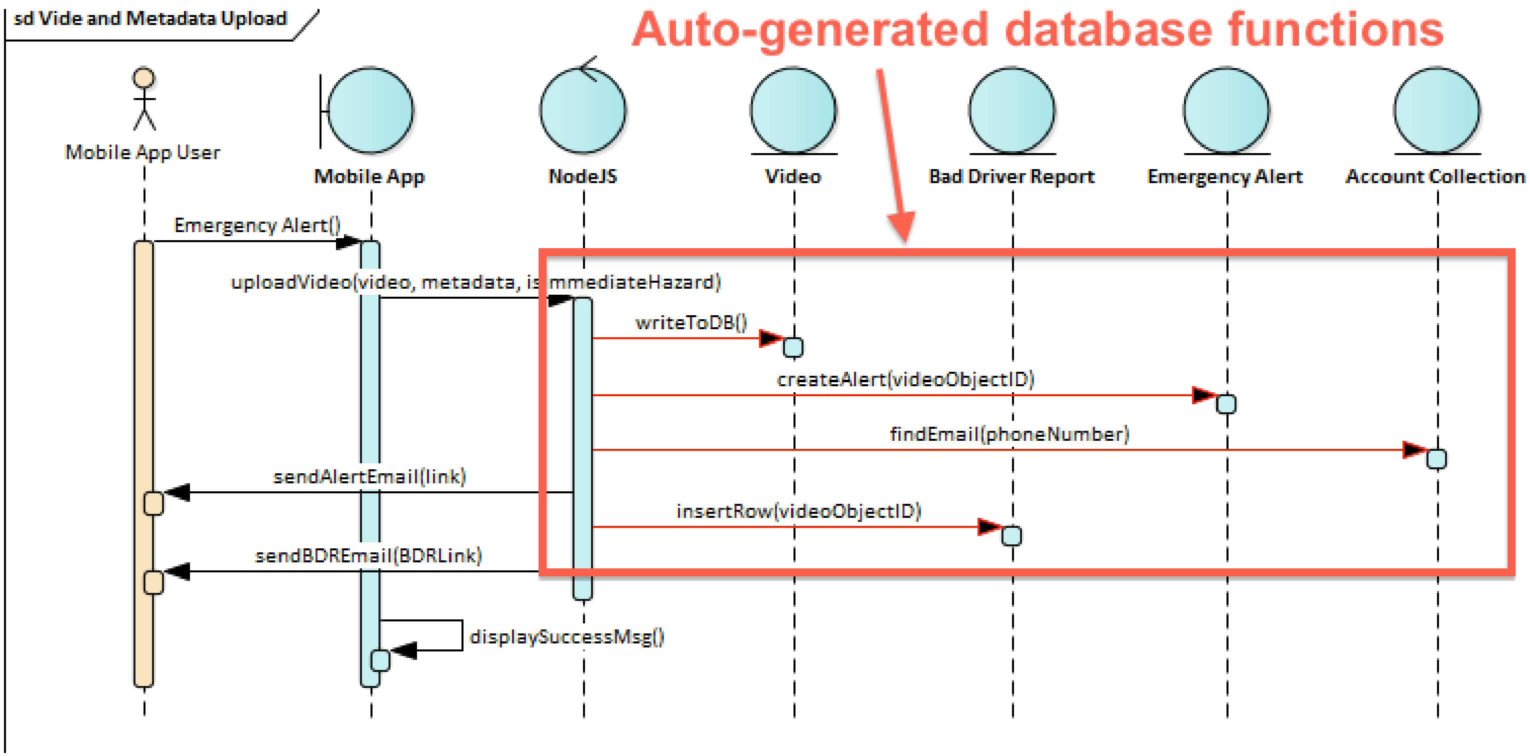
3 phase development – each phase takes about a month

- Inception – Model the problem domain and make it executable
- Phase 1 – Prototype to discover requirements
- Phase 2 – Model behavior to elaborate requirements
- Phase 3 – Acceptance test against requirements



PARALLEL AGILE

Database access code doesn't get written manually



in round numbers this might be 20-40% of your code

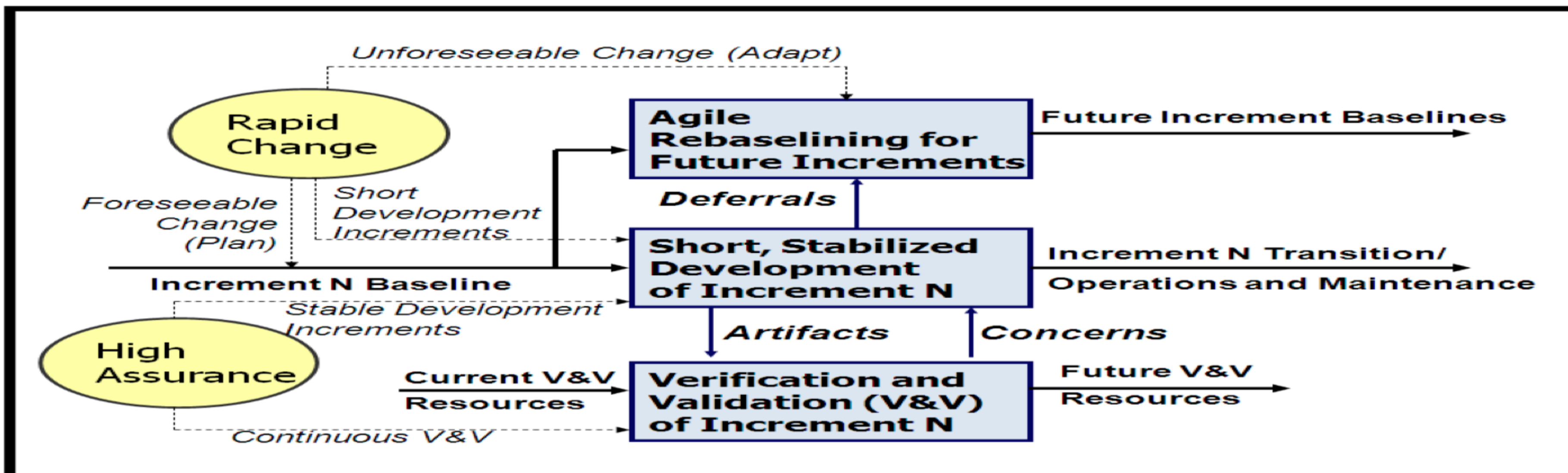
PARALLEL AGILE

– **Current status**

- 2014-2015 Location Based Advertising (75 students)
  - Implemented commercially; discontinued due to low sales
- 2015 Picture Sharing (12 students)
  - Experiment comparison with Architected Agile project
  - PA project faster, less effort; comparable performance
- 2016-2018 CarmaCam (75 students)
  - In LA-Metro experimental use for bus-lane monitoring
  - Several additional organizations, applications interested
- 2017-2018 TikiMan Go Game project (25 students)
  - Being prepared for commercial application
- Engaged in exploratory PA-based Mongo/Node replacement for 30 year old legacy database system

Large Scale PA Critical Success Factors

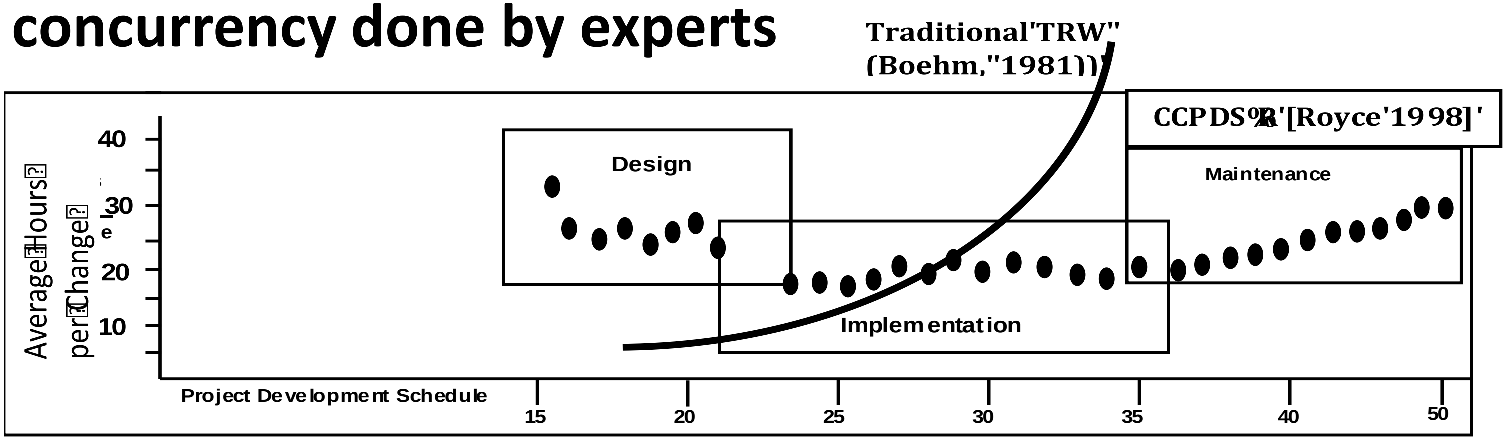
- Three Team approach; similar to Bosch ART approach
  - Agile Rebaselining: Keeper Of The Project Vision/Architecture
    - USC: Rosenberg: Ensure MVC compliance, rainy-day use cases
    - TRW: Systems Engineering team; Handle all concurrency
  - Developers and Product Owners:
    - Rapid concurrent development
  - Independent Verification and Validation
    - Continuous across development



SYSTEMS ENGINEERING
Research Center

TRW Large-Scale PA Experience

- Walker Royce: 1-million SLOC Command-Control System
- Extensive early architecture and risk resolution; all concurrency done by experts



  - 75 sequential-Ada programmers; Executing Arch. Skeleton
- Neil Siegel: several even-larger systems
  - Very high productivity; low error rate
  - Proof of value: worse productivity, error rate when new customer forced traditional approach; full productivity resumed when original approach resumed

History of COSYSMO Models

COSYSMO 1.0
Valerdi, 2005

- Identifies form of model
- Identifies basic cost drivers
- Identifies Size measure

With Reuse
Wang et al, 2008

- Adds weights to Size elements, reducing net Size in the presence of reuse

Req'ts Volatile
Pena, 2012

- Adds scale factor based on requirements volatility

For Reuse
Wang et al, 2014

- Adds weights to Size elements, reducing net Size when artifacts are only partially completed

Sys of Sys
Lane et al, 2011

- Allocates SE effort to SoS and constituent systems. Adds effort multiplier when in the presence of system-of-systems.

COSYSMO 3.0
Alstad, 2018

- Integrates features of previous models

COSYSMO 3.0
Top-Level Model

$$PH = A \times (AdjSize)^{E} \times \prod_{j=1}^{15} EM_j$$

Elements of the COSYSMO 3.0 model:

- **Calibration parameter A**
- **Adjusted Size model**
  - **eReq submodel, where 4 products contribute to size**
  - **Reuse submodel**
- **Exponent (E) model**
  - **Accounts for diseconomy of scale**
  - **Constant and 3 scale factors**
- **Effort multipliers EM**
  - **13 cost drivers**