A Systems Engineering Perspective of Aspect-oriented Software Architectural Analysis

Phillip Schmidt, Ph.D.
Phillip.P.Schmidt@aero.org

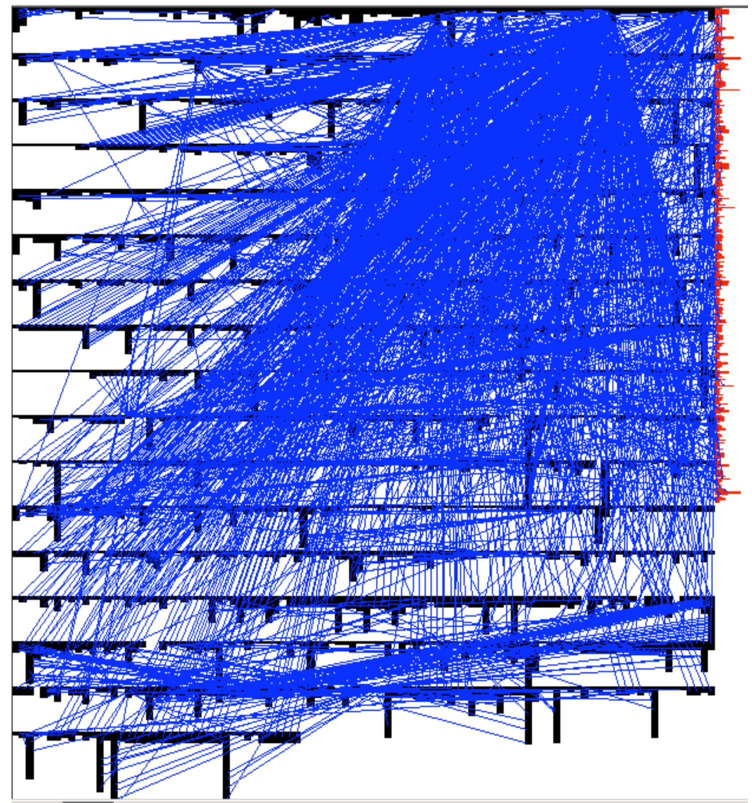
THE AEROSPACE CORPORATION

Agenda

- Our World
- Why a Systems Engineering Perspective?
- REACT
- Architectural Representation Challenges
- Evolving REACT
- Closing Remarks

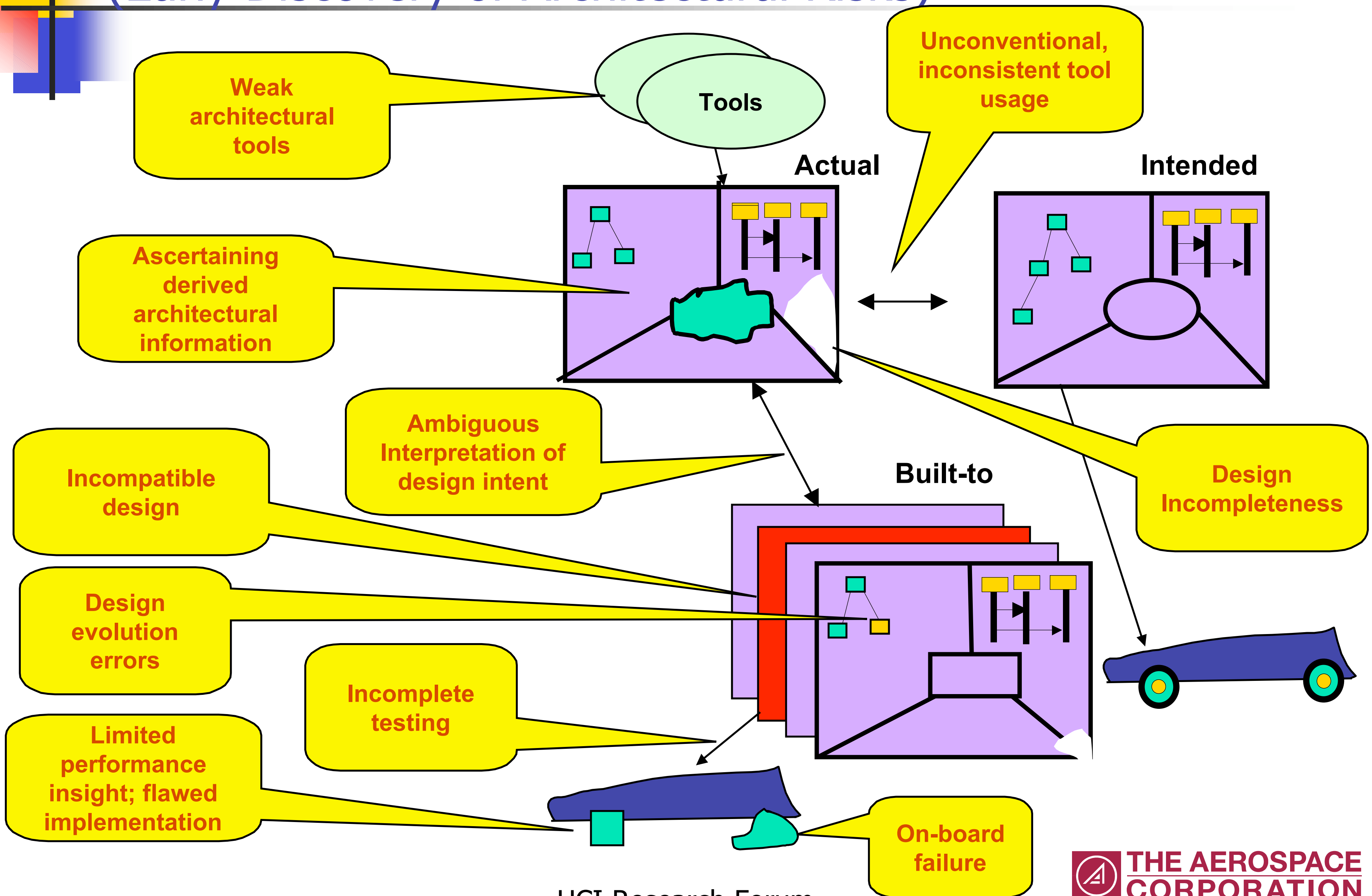
Our World is Rocket Science!

- Space system architectures growing increasingly complex
  - Highly interdependent legacy subsystems
  - Manual inspection of hardcopy designs ineffective in finding subtle design flaws
  - Increasingly difficult to make technical tradeoff decisions based solely on qualitative judgments (e.g. within Integrated Product Teams)
  - Architectural representation issues, object-oriented design technologies applied to legacy RT embedded systems not well understood
- Space system architectures exhibit pressure to evolve
  - Desire to improve performance, functionality, and program success
  - New environments
  - New services
  - New contexts
- Complexity and evolution raise risk



How do we manage architectural risk?

Challenges
(Early Discovery of Architectural Risks)



Why a System Engineering Perspective?

- Disconnect between Vision and Reality:
  - Vision: Architecture is central to supporting program evolution
  - Reality: Software architectural representations often incomplete and inconsistent
- A systems engineering perspective is needed to recognize and deal with the disconnect
- Architecture is more than
  - what UML is today
  - what Aspect-oriented programming is today (and will likely become)
  - questions about code
- Architectural representation challenges await
- Aspect-oriented architectural analysis is being used to tackle these challenges

Real-time Embedded Architecture-Centric Testbed (REACT)

- Architecture-Centric
  - Recognize importance of architectural representation
    - Many forms
    - Frequent access
  - Early discovery/feedback
- Aspect-Oriented Architectural Assessment
  - Architectural development exhibits concerns that cut across object decomposition boundaries
  - Support for automated management of concerns

Architecture-Centric

- Receive contractor-provided architecture artifacts
  - Unified Modeling Language (UML)
  - Other electronic representations
- Automatically extract architectural information
- Conduct architectural assessments
  - Prior to code development
  - Static Assessment
    - consistency/completeness
    - Compare "as-designed" to "as-built" representations
  - Dynamic Assessment
    - Focus on critical execution issues (synchronization, priority tasking, sizing)
    - Create simulations of well-formed models
    - Understand logical execution behavior of architecture
    - Refine/re-parameterize models
- Work closely with program office/contractor
- Work closely with UML vendors

Aspect-Oriented Architectural Analysis

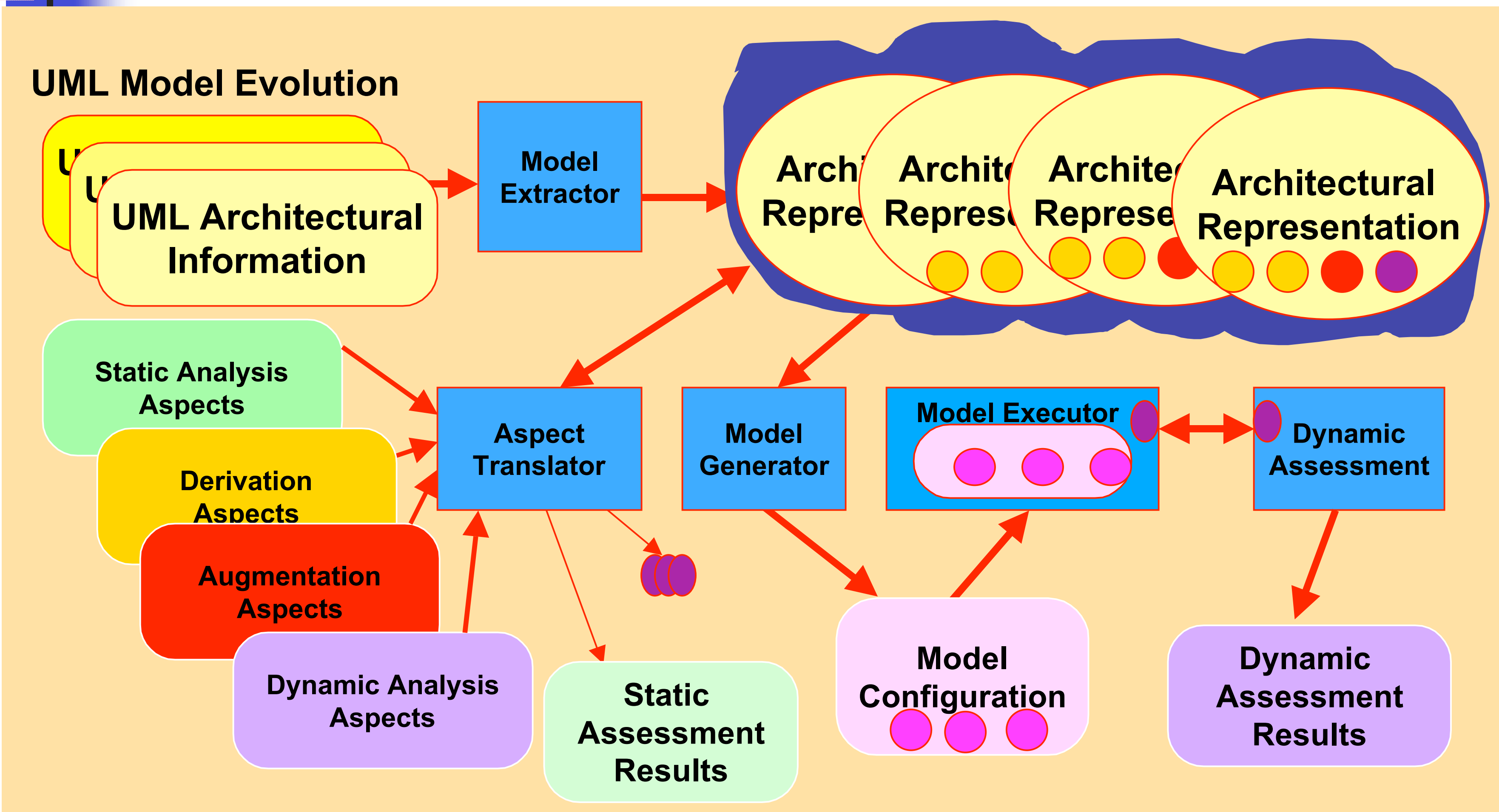
- Idea: Apply aspects over UML architectural domain

AOP	AOAA
Leverage expression of cross-cutting concerns	Leverage expression of cross-cutting concerns
Programming language domain (e.g. Java)	Architectural domain (e.g. UML and other artifacts)
Solutions architecturally intrusive (completeness)	Architecturally non-intrusive; separable via simulation
Address dynamic, execution impacts	Address static or dynamic aspects

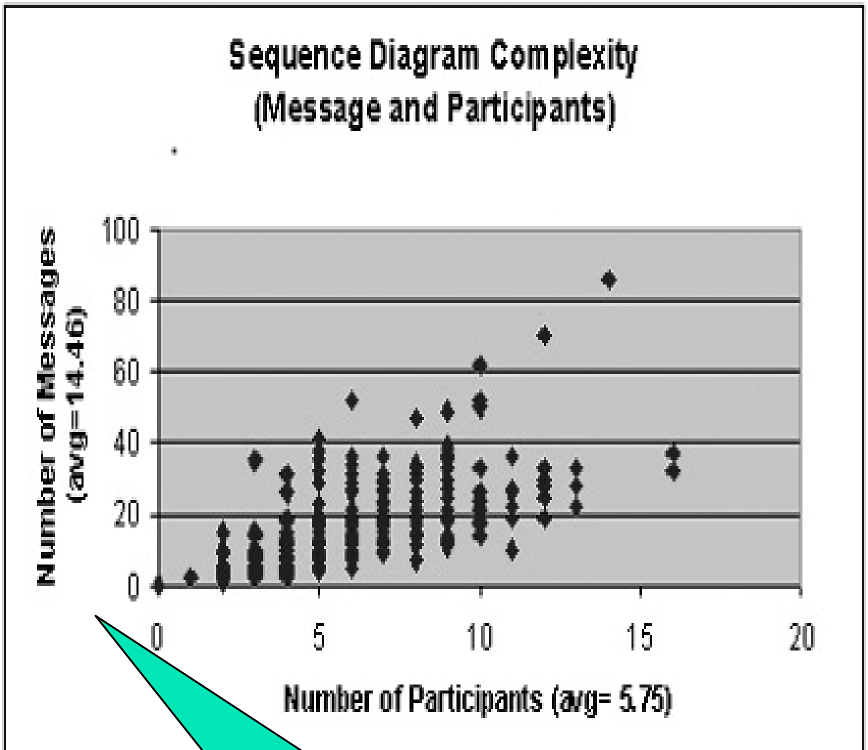
Architectural Aspect Types

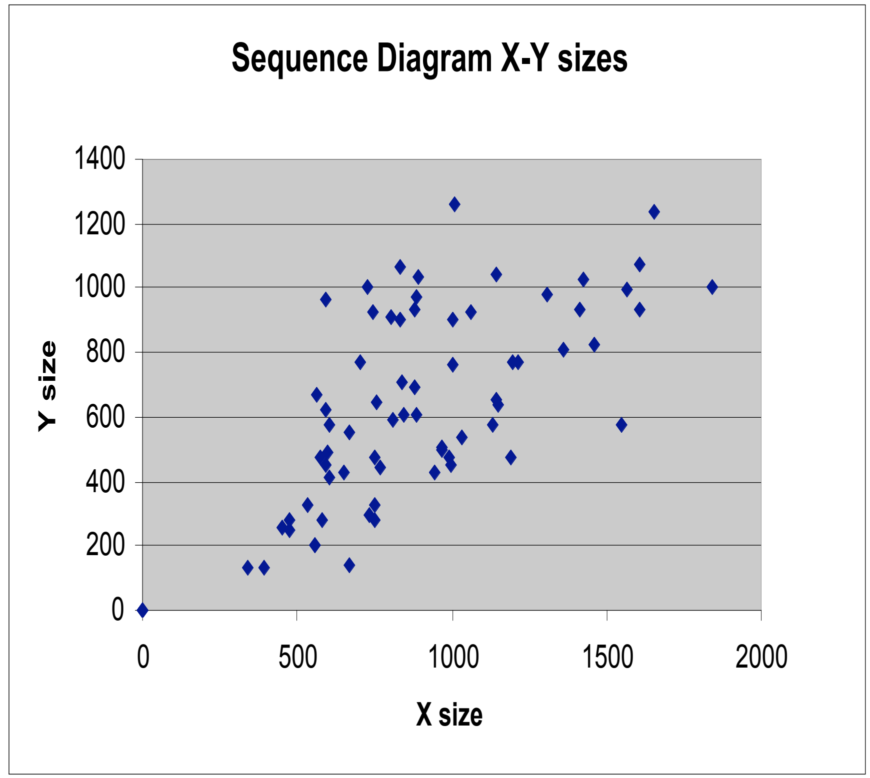
Aspect Type	Description	Example
Static Analysis Aspects	Perform integrity, consistency checks over UML space	Find all examples of destroy object usages
Derivation Aspects	Derive new or customized architectural information from UML space	Collect all event related information
Augmentation Aspects	Add new architectural informational detail	Supply model information based on ICDs, other analysis
Dynamic Assessment Aspects	Define cross-cutting concerns that need to be monitored	Log all raised exceptions; evaluate pre/post conditons

Real-time Embedded Architecture-Centric Testbed (REACT)
Aspect-Oriented Architectural Assessment



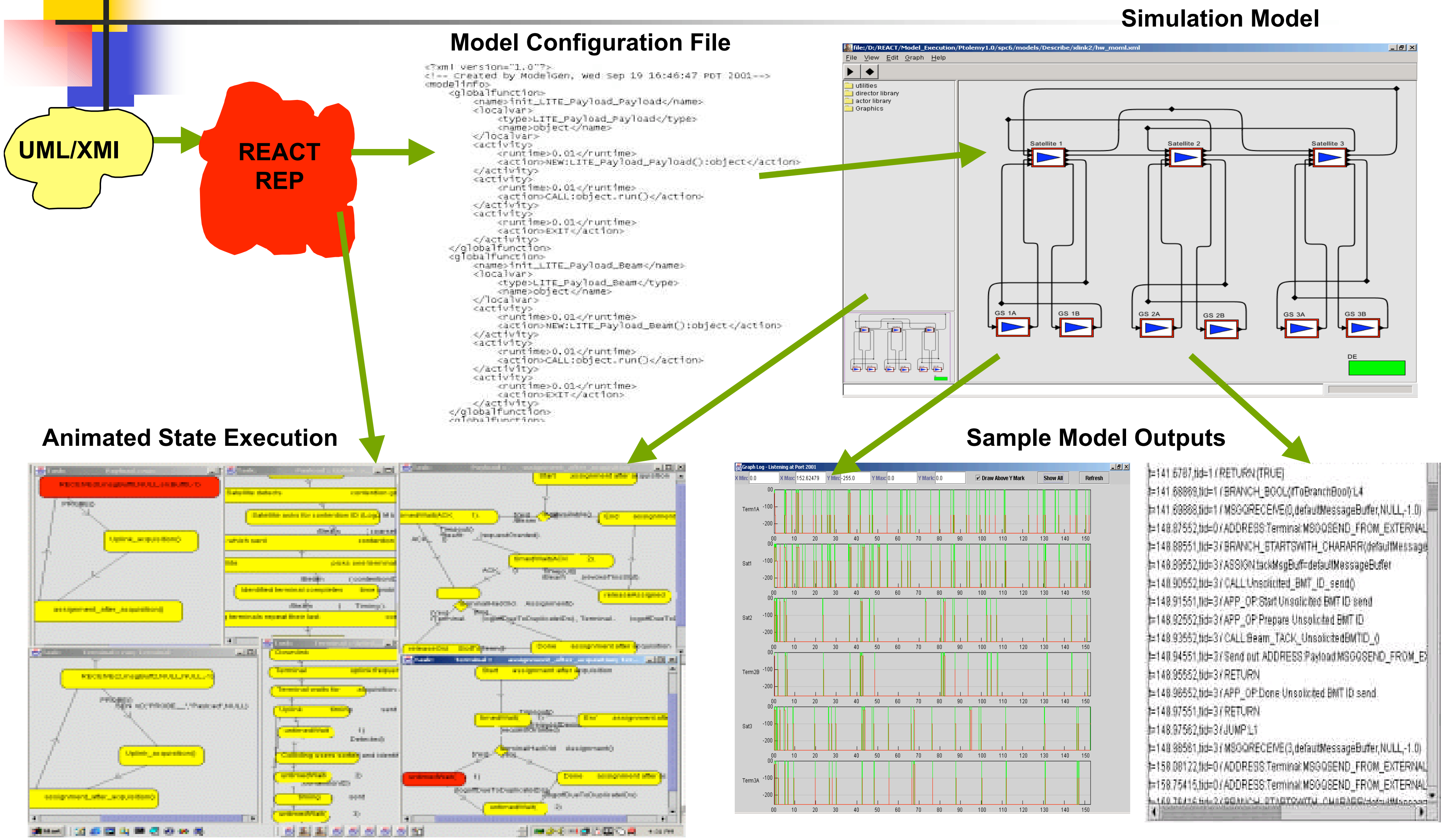
Aspects useful in exploring quality concerns





Message interaction sets too large for manual inspection

Dynamic Architectural Analysis



UCI Research Forum

THE AEROSPACE CORPORATION

There still are problems…

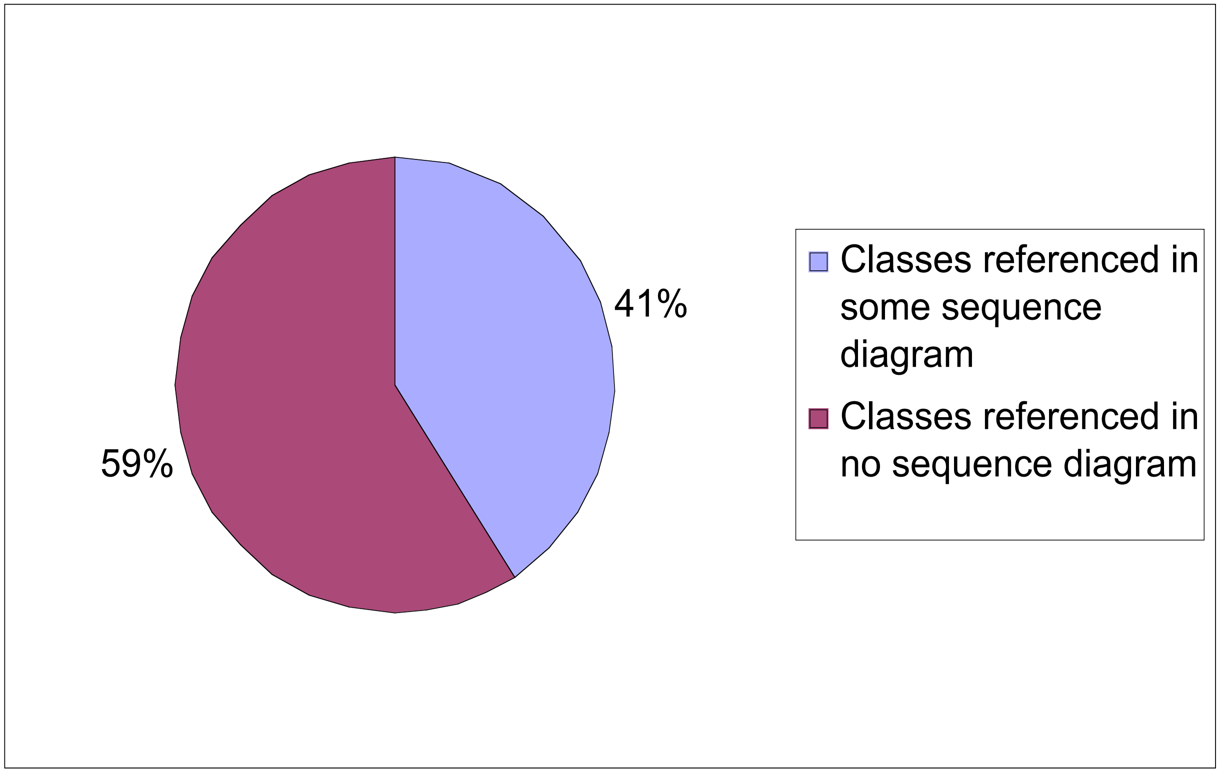
Problem Areas	**Solution Approaches**
Human factors, Architecture-Centric philosophy not always embraced	Improve trust, education, tools, methodologies, research
UML Usages	UML profile, improved architectural semantics
Inconsistency	Early discovery, Static analysis
Behavioral incompleteness	Augmentation, auto-generation, re-parameterization
Dynamic Assessment	Multi-level modeling techniques
Architectural Evolution, Cross-cutting concern analysis, etc	Better model representation/analysis techniques. Aspects

Ignoring these does not reduce architectural risk

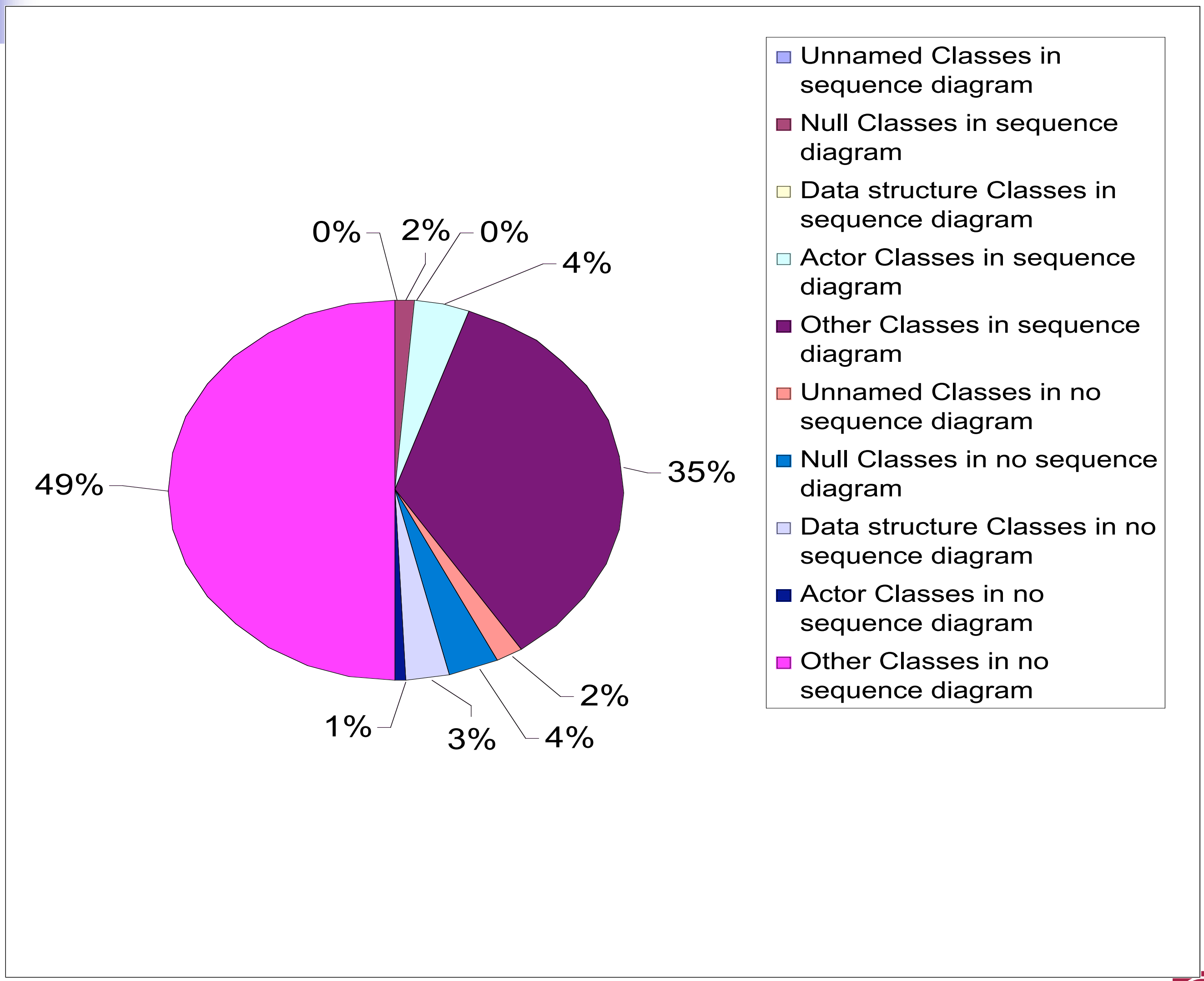
Different UML Usages

Focus	UML Artifacts	PIM	PSM
Conceptual system-level models (goals, objectives, system dependencies, constraints)	High-level sequence diagrams High-level state/activity diagrams Class/actor as subsystems Role relationships between components	**X**	x
Requirements analysis and traceability (reqt ids, subsystem, build, test info)	**Use case/functional requirement descriptions (nominal, alternative, exception, preconditions, postconditions, triggers)**	**X**	x
Architectural/detailed design Level (active/passive objects interfaces, tasks, OS models, concurrency,)	**Class diagrams as SW classes** **Detailed sequence diagrams (messages/methods, class participants)** **State behavior (class, method)** **Deployment info**	x	**X**

REACT Example:
Class Coverage in Sequence Diagrams

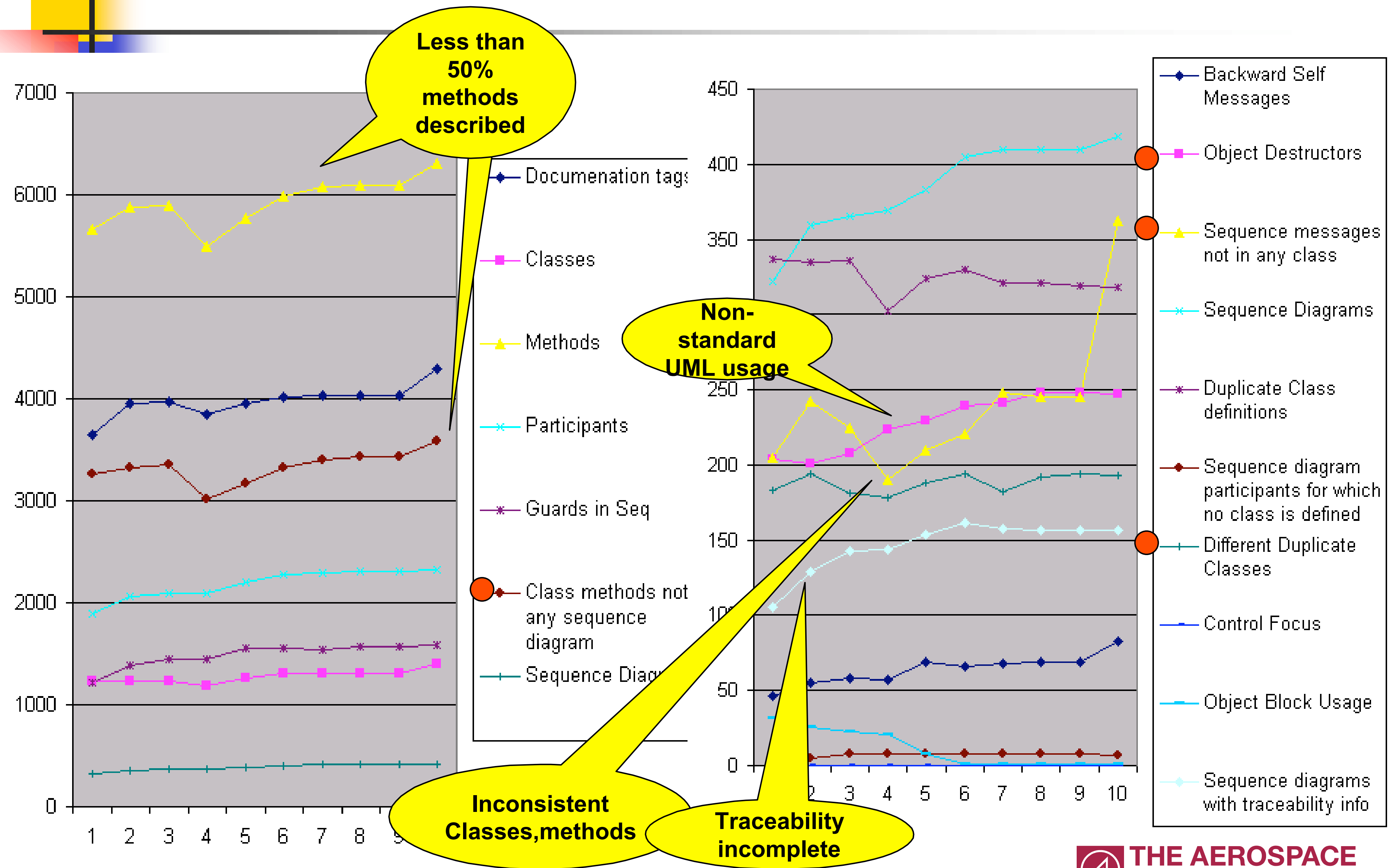


Usage of Class Diagrams



REACT Early Discovery Example:

Consistency and Completeness



Dynamic Assessment

- Goal: Perform dynamic assessment when model behavioral information is missing
- Approach:
  - Multiple levels of modeling abstraction
  - Augmentation aspects
  - Monitoring aspects

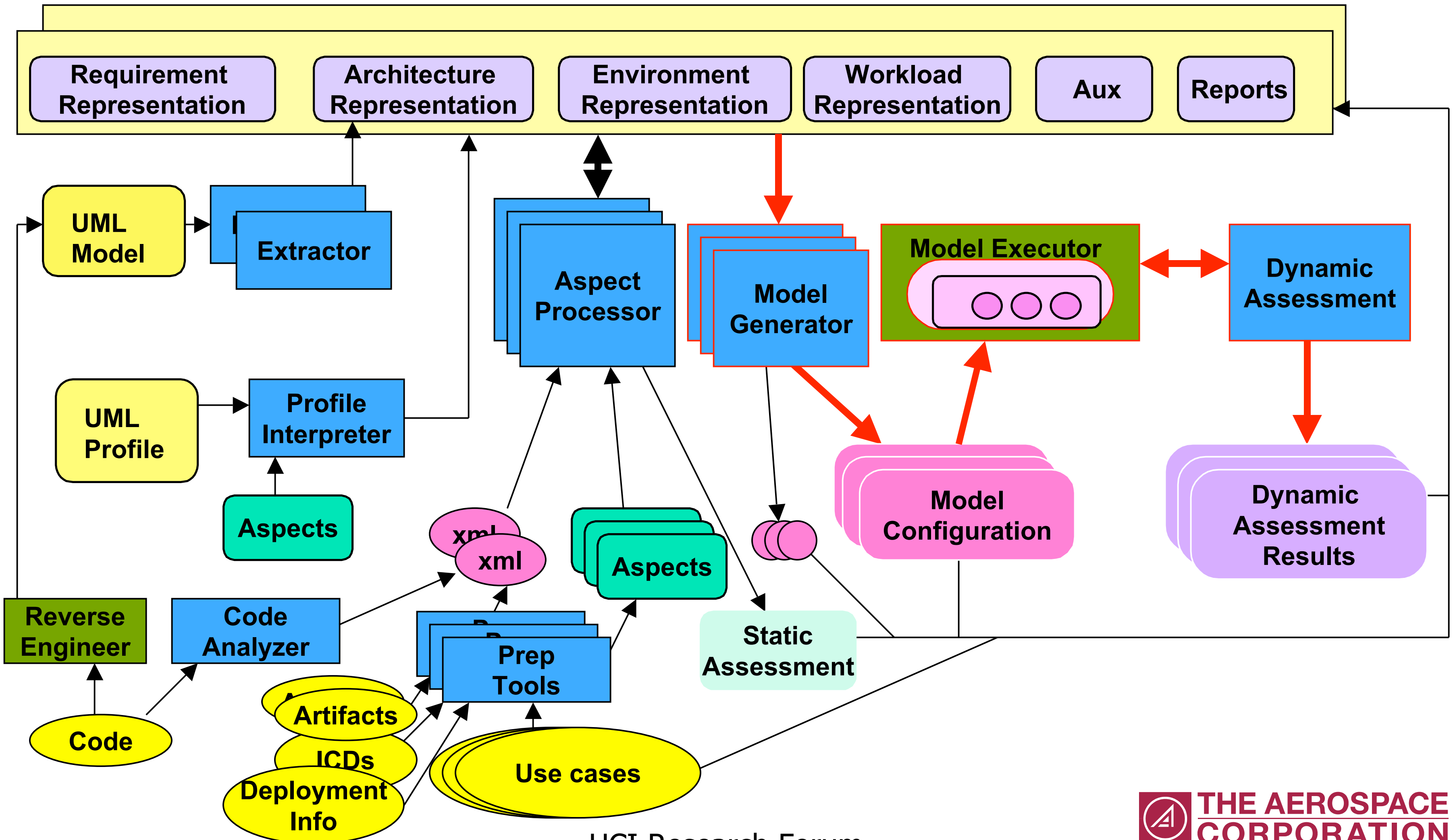
Architectural Evolution

- Representations must support frequent change (mandatory/optional components)
- Not all features will be preplanned and separable
- Need to look backward, forward, and elsewhere! (e.g. old design decisions, new usage scenarios, other ICDs, changing requirements)
- Expand features to study concerns we don't want! (e.g. design conflicts, deadlocks, unreachable states)
- Architectural complexities/dependencies will make feature interactions difficult to manage
- Separation/integration of multiple UML models
- Any given OO decomposition will eventually be reexamined
- There are cross cutting concerns that the programming domain alone cannot answer (e.g. version impacts, requirements evolution changes, workload)

Evolving REACT

- Improve Architectural Representation
- Improve Assessment Techniques

Expanding Architectural Representations



Expanding Assessment Techniques

- Develop tools/techniques to improve context and semantics
  - XML schemas represent/share architectural artifacts
  - Support augmentation from various sources
  - Support interpretation aspects (e.g. UML profiles of use)
- Augment representations with parameters derived from reverse-engineered code
  - Capture missing behaviors to improve evolution success
- Manage planned scenarios as analyzable use cases
- Manage planned features as aspects over entire representation space
  - Dependencies too difficult otherwise
- Move toward automating analysis and aspect-oriented impact analysis
- Develop architectural analysis techniques to discover design patterns and refactoring opportunities

Closing Comments

- The holy grail of architecture is not efficient software code generation but managing architectural risk during its evolution
- A systems engineering perspective supporting architectural assessments and impacts to change is desired
- Architecture is a core asset that goes beyond UML and AOP.
- Architectural representation challenges remain
- UCI is a meeting the challenge!

Backup Charts

References

- **Scenario-based**
  - R. Kazman, G. Abowd, L. Bass, P. Clements, " Scenario-Based Analysis of Software Architecture," *IEEE Software*, 13 (6):47-56, 1996
  - R. Kazman, M. Klein, M Barbacci, T. Longstaff, H Lipson, J. Carriere, "The Architecture Tradeoff Analysis Method," in Proceedings of the 4th International Conference on Engineering of Complex Computer Systems (ICECCS98), Monterey, CA, IEEE CS Press, pp68-78, 1998
  - P. Bengtsson, N. Lassing, J. Bosch, H. vanVliet, "Analyzing Software Architectures for Modifiability," May 2000
- **Feature-oriented**
  - M. Svahnberg, J. Van Gurp, J. Bosch, "On the Notion of Variability in Software Product Lines" *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA 2001),* pp 45-55, August 2001
- **Architecture-centric Design**
  - T. Mens, C. Lucas, P. Steyaert, "Supporting Disciplined Reuse and Evolution of UML Models," *First International Workshop on UML*, Mulhouse, France, June 1998
  - Xadl, an XML architecture description language http://www.isr.uci.edu/projects/xarchuci/

References (cont'd)

- **Aspect-oriented Programming**
  - See *Communications of ACM*, October 2001, Vol 44, No 10.
- **Aspect-oriented Architectural Analysis**
  - P. Schmidt, R. Duvall, G. Mulert, J. Milstein, J. Rivera, "Aspect-Oriented Architectural Analysis using Multi-level Modeling of Complex Systems," *Proceeding of 2003 International Information Resources Management Conference*, May 2003
- **Maintenance/Product Line Studies**
  - J.Bosch "Product-Line Architectures in Industry," *Proceeding of the 21st International Conference on Software Engineering*, Nov 1998
  - J. van Grup, J. Bosch, "Design Erosion: Problems and Causes," *Journal of Systems and Software*, November 2001.
  - M. Lindvall, K. Sandahl, "How well do Experienced Software Developers Predict Software Change?", *Journal of Systems and Software,* vol 43, no 1, pp 19-27, 1998
  - M. Lindvall, M. Runesson, "The Visibility of Maintenance in Object Models: An Empirical Study," *Proceedings of International Conference on Software Maintenance*, Los Alamitos, IEEE CS Press, pp 54-62, 1998
  - B. Lientz, E Swanson, *Software Maintenance Management*, Reading, MA, Addison-Wesley, 1980.

Definitions

- Architectural variability, refers to the ability to identify and flexibly reshape aspects of an architecture
  - Aspects identify points of variation

- Program evolution refers to the ability of an architecture, over its lifecycle, to undergo change

Augmentation Aspects

- Example: Initially model missing information as a "black box"
  - An aspect identifies
    - Area/context of interest (e.g. methods with no state behavior)
    - Some action to be taken (associate some default black box action state with that method)
  - Later another aspect could replace/revise the black box behavior
- Example: Identify all COTS tool interfaces

Monitoring Aspects

- Monitor defines an action to take and the condition under which to enable it.
- Currently monitoring is independent of system under study. E.g. monitoring does not force adaptive behavior
- Augmentation aspects can tag areas and enable monitoring. E.g. All interrupt handler methods.
- Monitoring can provide directives to the simulator (e.g. report Task msg queue size)

Multi-Level Modeling Types

- Method-level Modeling
- Participant-level Modeling
- Use-case level Modeling