

Lesson Learned in the Specification/Development of Object-Oriented Space Ground System Architectures

James A. Hager

SRI International / Penn State University

jhager@scpa.sri.com / jah14@psu.edu

Unclassified

Brief Description of System

- What it is
 - Unified ground architecture for multiple sites
 - Tasking reception, mission planning, mission execution, mission assessment, reporting, command and control
 - Interface to collectors, processors, and external users

Brief Description of System

- Scope of System
 - 1.5 M Object-Oriented LOC
 - OO interfaces to ~1.5M for new collectors
 - OO interfaces to Midas2K for new processors
 - Encapsulators for legacy collectors and processors
 - 400 developers from 4 contractors at 4 locations

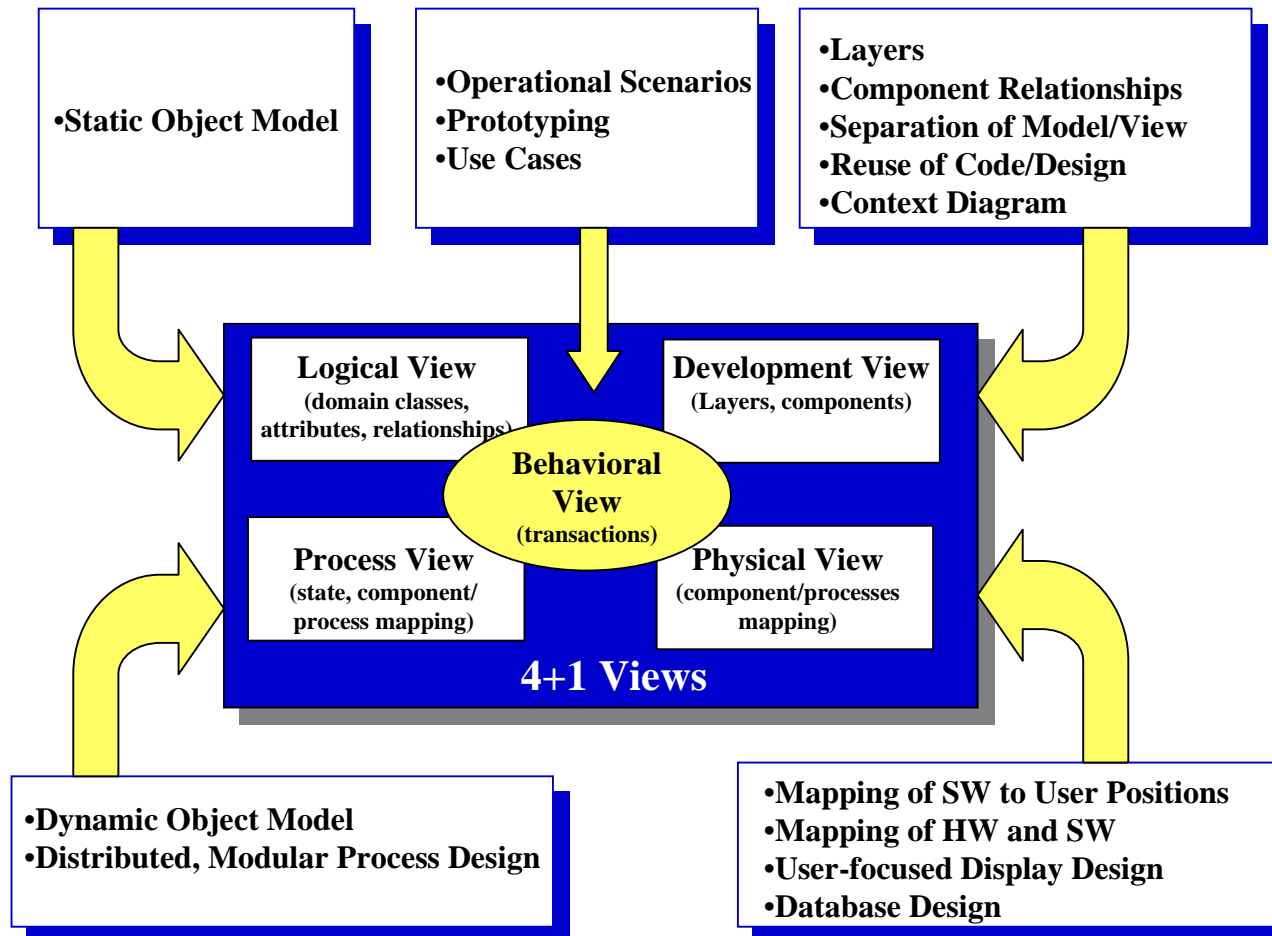
Architecture Ground Rules

- Based on a fundamental re-look at required “behavior” of SIGINT system
- Standardized collector and processor interfaces at all sites
- Common operations “look and feel” (to the extent reasonable) at all sites
- Maximum reuse of “common architecture elements”
 - Layered software architecture, Common COTS-based infrastructure, Reusable applications frameworks, Common services
- Encapsulation of legacy systems within fundamental architecture
- Evolutionary migration from current architecture to desired architecture features

Why OO now?

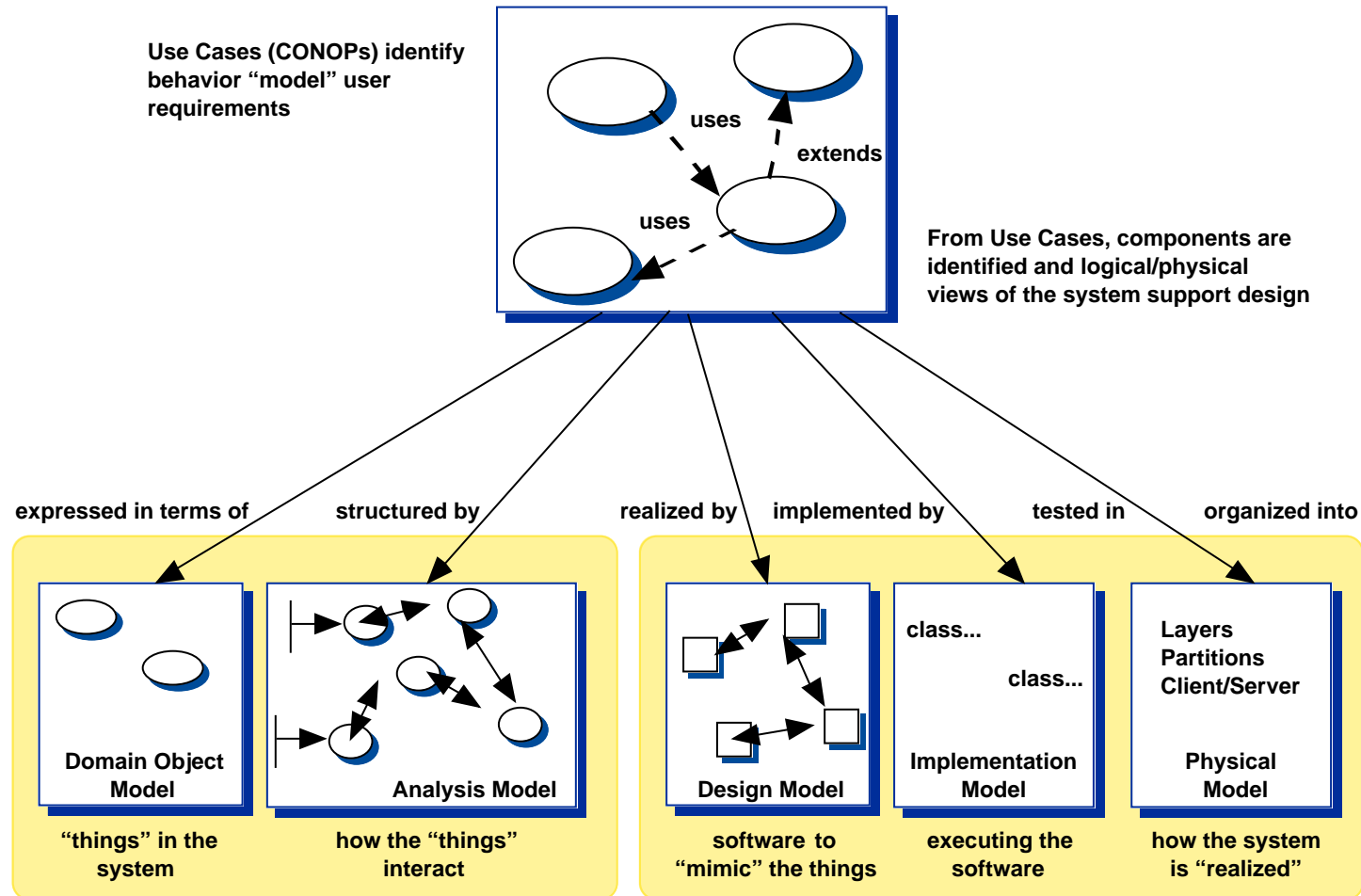
- Inherent support for fundamental re-architecture goals
- Maturing methodology and OO toolset support
- Maturing technology support base (e.g., OODBMS, GUIs, middleware, distributed computing)
- Emerging COTS standards
- Critical mass of available COTS products meeting the standards

Maturing Process View



Unclassified

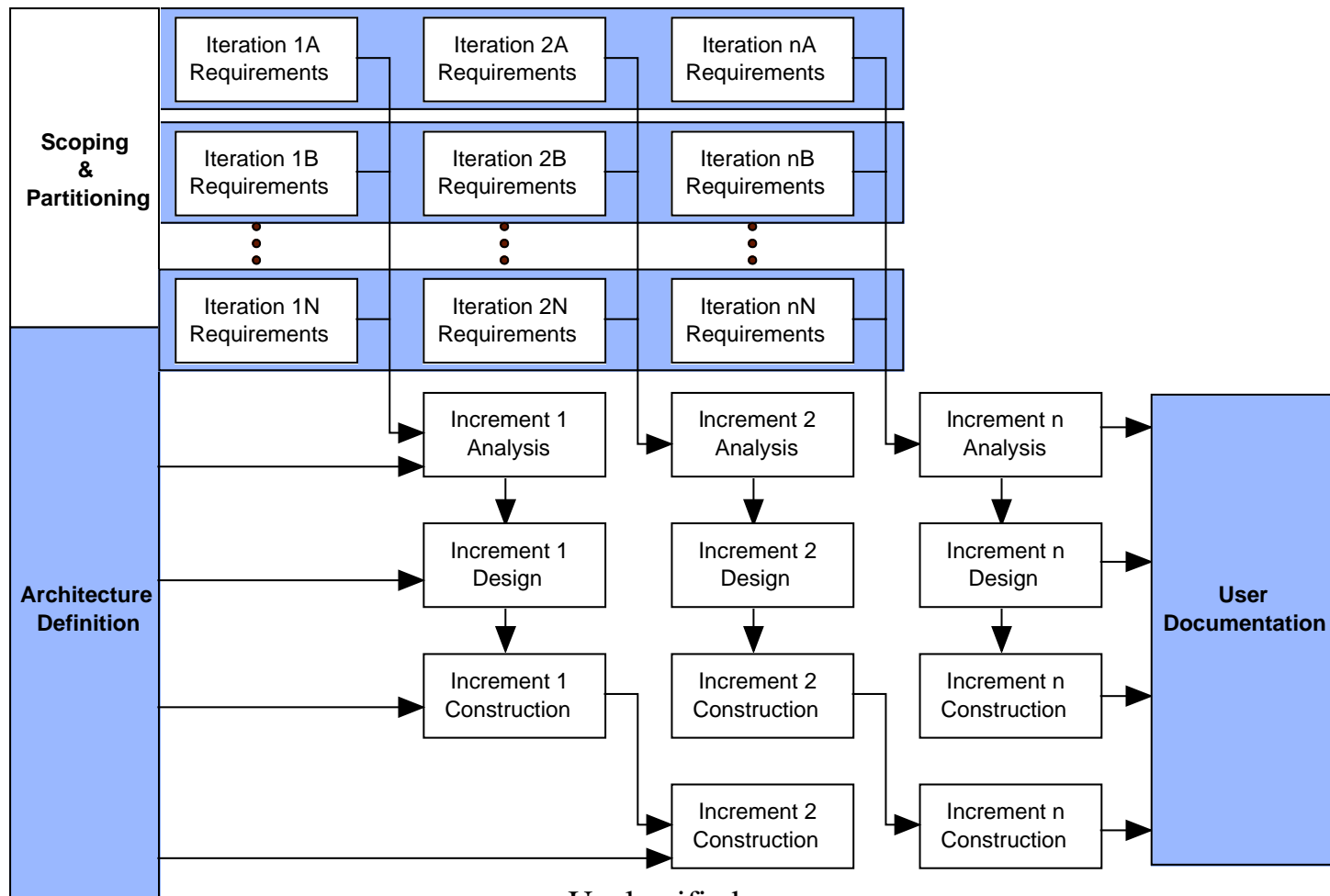
Maturing Process View OO Model Development



00498

Unclassified

Maturing Process View OO Development Lifecycle Stages



Unclassified

....Some Lessons ~~Learned~~
Identified

Unclassified

Key OO Observations

- OO is an enabling technology, not an End Unto Itself....It's just as easy (maybe a little easier) to fail with OO as it is with traditional structured analysis/design approaches

OO Technology

- Effective in managing system complexity BUT it is not a PANACEA
 - use of OO technology does not guarantee a successful project
 - Structured approaches offer (and in the hands of good designers deliver) many of the same promises
 - Keys to successful projects are still well-defined, understood and used program plans, requirements, and development processes
 - Using OO technology does not reduce the need for sound systems engineering

OO Component Technologies

Software Technologies

- Languages
C++, JAVA, SmallTalk
- OO DBMS
- Middleware
- Distributed Computing

Design Practices

- Design Patterns
- FrameWorks
- Encapsulation/abstraction
- Partitioning/Layering
- Information Hiding
- Standards

Processes -

Advanced Tool Support

- Booch, OMT, UML
- STP, Rose, RTM, CM
- Lifecycle
Spiral, Iterative, Incremental

Metrics

- OO Measurements-
depth of inheritance, number of children, cross-class category interactions, complexity, multiple inheritance, size, volatility, progress, staffing, training.

**Object Oriented
Methodology**



Unclassified

Architectural Beauty

-Should not be in the eye of the beholder

Architecture Policies

- What they are
 - Technical architecture basis for evaluating a system architecture
 - Capture sound design principles as formal policy for enforcement
 - Identify expectations of benefits and quality criteria
 - The basis for developing metrics to evaluate architecture
 - Cover major areas: abstraction, layering, partitioning, information hiding, encapsulation, external interfaces, legacies, patterns, frameworks, reuse, standards, COTS, persistence

Architecture Policies

Sound Design Principles

- *Reusable Design Patterns* - Gamma, et al
- *Reusable Frameworks* - Goldsmith, Taligent; Stroustrup, ATT Bell Labs
- *Encapsulation, Abstractions* - Stroustrup, ATT Bell Labs
- *Partitioning, Layering* - Booch, Rational
- *Layering, Abstractions* - Rumbaugh, Rational
- *Information Hiding, Change Isolation* - Parnas, Hager, et al NRL
- *Standards - DoD Technical Architecture Framework for Information Management (TAFIM)*

Architecture Policies

- What we've learned
 - Documented basis for evaluating “beauty” of system architecture
 - Compels developers to focus on architecture, not design details
 - Provides rational basis for a metrics program
 - Policies are candidate additions to DOD system acquisition models, TAFIM Extension Models

Architecture Policy Principal Benefits

Principal Benefits	Architecture Policy												
	External Interface	Legacy	Abstraction	Layering	Partitioning	Encapsulation	Information Hiding	Patterns	FrameWorks	Standards	COTS	Persistence	Reuse
Confinement of Change	X	X	X	X	X	X	X	X	X				
Reduced Implementation Dependencies	X	X	X	X	X	X	X						
Reduced Interface Complexity	X		X	X	X	X	X	X	X				
Reduced Operational Complexity	X	X											
Extensibility	X		X	X	X	X	X	X	X	X	X		
Reduced Effort to Maintain	X	X	X	X	X	X	X	X	X	X	X		
Improved User Productivity	X												
Efficient Usage of Training Resources	X												
Improved Quality of Training Materials	X												
Deferred Obsolescence		X						X	X	X	X		
Separation of Concerns	X	X	X	X	X	X	X	X	X				
Reliability			X	X	X	X	X	X	X	X	X		X
Design Modularity/Weak Coupling			X	X	X	X	X	X	X				
Reuse Support		X	X	X	X	X	X	X	X	X	X		X
Conceptual Simplification			X	X	X	X		X	X				
Design/Development/Testing Efficiency	X	X	X	X	X	X	X	X	X	X	X	X	X
Incremental Build Support			X	X	X	X	X	X	X		X		X
Horizontal and Vertical Scalability			X	X	X	X	X	X	X	X	X		
Tradeoff Support			X	X	X					X	X		X
Concurrent Development Support				X	X			X	X				
Portability			X	X	X	X	X	X	X	X	X	X	
Interoperability			X	X	X	X	X	X	X	X	X	X	
Risk Reduction	X	X	X	X	X	X	X	X	X	X	X	X	X
Interchangeability	X	X						X	X	X	X		
Design/Operational Flexibility			X	X	X	X	X	X	X	X	X		
Enhanced Performance											X	X	
Uniformity/Consistency		X						X	X	X	X	X	
Robustness				X		X	X	X	X	X		X	

00698

Unclassified

Metrics Program

- *....More important* than ever when evaluating OO developments by contractors and acquisition personnel more comfortable with functionally-oriented developments

Measurement Program

- What we've learned
 - Development contractors reluctant to expose measures of progress and quality...no surprises here!
 - OO metrics tend to be academically oriented and frequently focus on the irrelevant...misses mark
 - Development tool vendors had to be contracted to add appropriate technical metrics collection and reporting
 - Essential for understanding large, complex analysis and design models
 - Need to be coupled closely with architecture policy initiatives

Architecture Policy Quality Measures

Quality Measures	Architecture Policy												
	External Interface	Legacy	Abstraction	Layering	Partitioning	Encapsulation	Information Hiding	Patterns	FrameWorks	Standards	COTS	Persistence	Reuse
Coupling (cross boundary associations, inheritance, aggregations)	X	X	X	X	X	X	X	X	X		X		
Number of External Entity Interactions	X	X	X	X	X	X	X	X	X		X	X	
Interface Changes	X	X	X	X	X	X	X	X	X		X	X	X
Defect Analysis (number, density, effort, root-causal)	X	X	X	X	X	X	X						
Depth of Inheritance			X	X	X	X					X		X
Fan In			X	X	X	X							X
Number of Children			X	X	X	X							X
Cohesion in Methods			X	X	X	X	X	X	X				
Percentage Public/Private Data			X	X	X	X							
Access to Public/Private Data			X	X	X	X							
Cyclomatic Complexity			X	X	X								
Actual Complexity			X	X	X								
Module Design Complexity			X	X	X								
Integration Complexity	X	X	X	X	X	X	X	X	X				X
Global Data Complexity			X	X	X	X	X						X
Goal vice Action-path Analysis	X												
Number of Proprietary Service Implementations										X	X		
Number of Classes			X	X	X	X							
Number of Subclasses			X	X	X	X							
Total Methods/Class (Public/Private/Protected)			X	X	X	X							
Number of Public Methods/Class			X	X	X	X							
Number of Abstract Classes			X	X	X	X							X
Number of COTS Implemented Abstractions											X		
Number of LOC (new, modified, reused)	X	X	X	X	X	X	X	X	X		X	X	X
Number of Frameworks/accesses									X				X
Number of Patterns/Accesses								X					X

00598

Unclassified

System Architecture

- Start at the beginning, proceed until you come to the end, and then stop....

System Architecture

- What We've Learned
 - tendency for system-level architectural issues to get lost in the carnage of design - rampant forest for the trees issue
 - Reluctance to talk architecture (context diagrams, components, interfaces, services, layers/tiers/partitions, etc)....tendency to fixate on evolving design models

System Architecture

- What We've Learned (Cont'd)
 - Healthy appreciation for the complexity associated with generating meaningful SGS abstractions (this was difficult engineering!) , coupled with unwillingness of diverse acquisition organizations to pay the bill for abstractions
 - Encapsulating legacy collectors and processors with OO-to-other translators is an art.....blackhole for unresolved architecture decisions

System Architecture

- What We've Learned (Cont'd)
 - Difficulty getting developers to fully exploit evolving OO technology base (“apriori” trade studies)
 - guaranteed interfaces and robust services provided by CORBA
 - natural persistence provided by ODMG-compliant object databases
 -but at times a preponderance of “elephant-gun solutions”
 - OO technology may not be as important as other factors in reducing long-term cost and complexity of developing and maintaining systems

Architecture Model Development

- What we learned
 - Initial object models from domain-savvy non-OO developers were bad....functional creep
 - Initial object models from OO-savvy non-domain consultants were bad...expert witnesses
 - Object models that were good took nearly two years to emerge
 - Analysis needs to start with a robust context diagram, elaborated with use cases and scenarios
 - Partitioning the analysis object model for design and development must be based on cohesion/coupling measurement analysis (measurement support!)

Reference Models

-necessary but not sufficient for the development of portable, reusable, interoperable, maintainable,.....xxx'able systems

Technical Reference Architecture, Standards, Products

- What we learned
 - TAFIM-based reference architecture was necessary but not sufficient: a convenient framework in standard form
 - System standards (POSIX, TCP/IP, Motif, SQL, OQL, CORBA, etc) for software infrastructure were necessary but not sufficient for portability and interoperability (due diligence, development/usage standards)

Technical Reference Architecture, Standards, Products

- What we learned (Cont'd)
 - Selecting 20 - 30 COTS products to support a modern software run-time infrastructure was a major undertaking
 - Coordinating the maintenance and upgrade (to the patch level) of the run-time COTS infrastructure needs constant coordination
 - Enforcing a common development infrastructure on multiple contractors has both good and bad effects

Second Class Promotions

- Beware the second class citizens.....many second class citizens become first class players in OO developments

Example Second–First Class Citizens

- Configuration Management
 - lifecycle complexity (e.g. builds, iterations, increments)
 - dynamic models
 - multi-contractor component developments