33rd Annual INCOSE international symposium

hybrid event

Honolulu HI USA

Architecting Descriptive Models for MBSE

Ryan A. Noguchi, The Aerospace Corporation

Outline

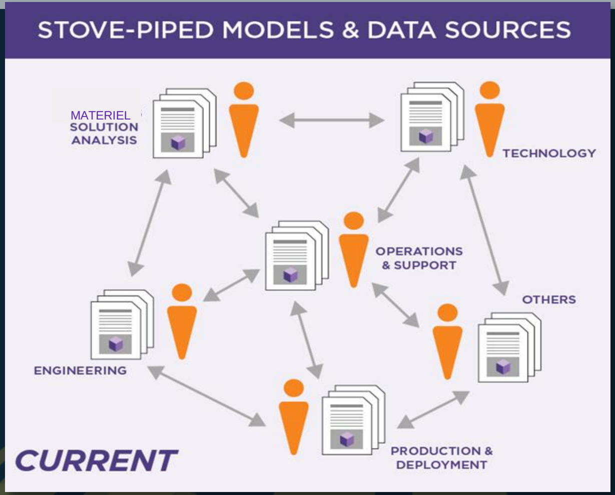
- Introduction – MBSE and the characteristics of descriptive models
- Foundations - Systems and software architecting
- The discipline of model architecting
- Model architecture patterns and heuristics
- Role and competencies of the Model Architect

Introduction – MBSE and the characteristics of descriptive models

Document-driven engineering to Digital engineering

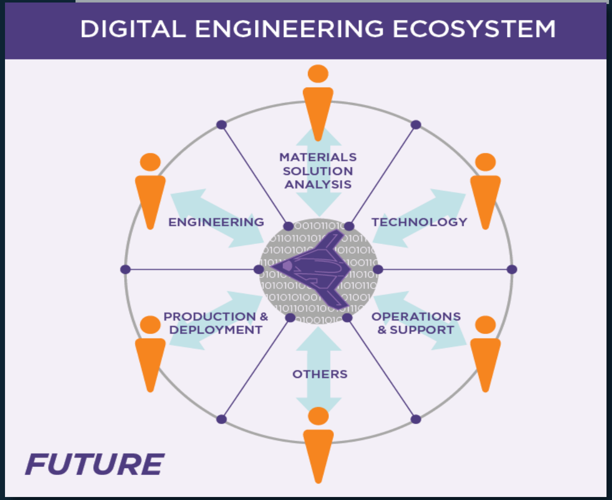
Document-Centric Culture

Document-driven engineering



Data/Model-Centric Culture

Digital engineering



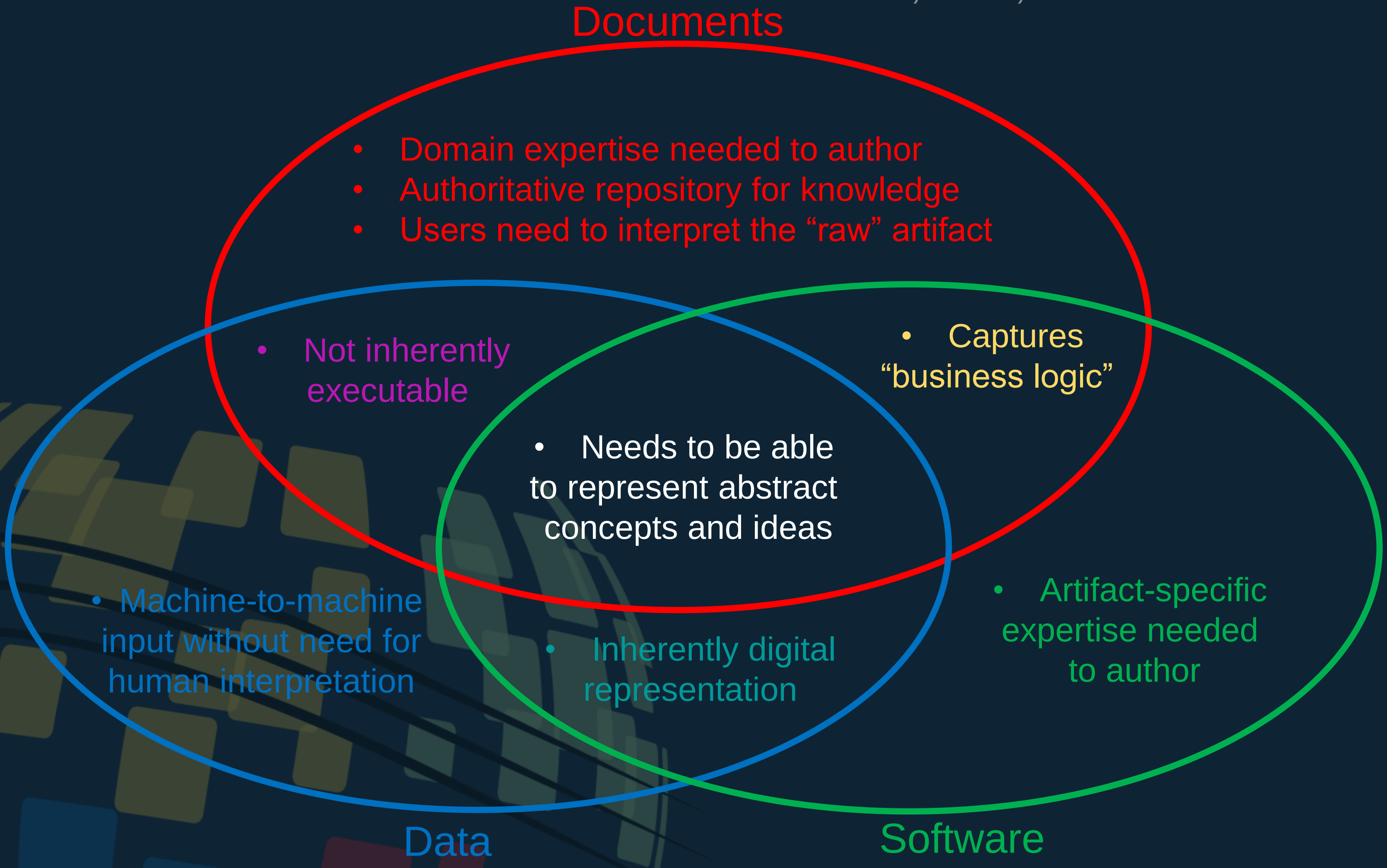
- Evolving knowledge is captured and exchanged via disconnected static artifacts (documents) and human-in-the-loop exchanges
  - *Often difficult to find the right documents*
  - *Documents are difficult to keep synchronized*
  - *Disconnects often discovered very late*
  - *Information exchanges are error-prone and time-consuming, sometimes completely missed*
  - *Misinterpretation is not uncommon*

- Evolving knowledge is captured in authoritative sources that are connected through federated sets of digital models
  - *Better and more efficient data management with greater consistency, integrity, fidelity, currency, timeliness*
  - *Faster and more reliable information exchange*
  - *Less-ambiguous representation reduces misinterpretation*

SE must transition to MBSE to enable this Digital Engineering transformation

Introduction – MBSE and the characteristics of descriptive models

Descriptive models have similar characteristics to documents, data, and software


Documents
• Domain expertise needed to author
• Authoritative repository for knowledge
• Users need to interpret the "raw" artifact
• Not inherently executable
• Captures "business logic"
• Needs to be able to represent abstract concepts and ideas
• Machine-to-machine input without need for human interpretation
• Inherently digital representation
• Artifact-specific expertise needed to author
Data
Software

This hybrid nature of models drives the need for a hybrid approach to developing them

Foundations – Systems and software architecting

Thinking of descriptive models as a system

	The "Real" System	The "MBSE" System (Descriptive models)
Functional Requirements	• What must it do?	• What questions must it answer?
Performance Requirements	• How well must it do it?	• How well must it answer them?
Structure	• What system components comprise it? • How are those components connected?	• What model projects comprise it? • How are those model projects federated?
Requirement Traceability	• Requirements trace to higher requirements	• Questions trace to higher questions—and to the decisions they inform
Requirement Allocation	• Requirements are allocated to sets of system components	• Questions are answered by sets of federated model components (and possibly other data)
Requirement Verification	• Test events verify requirements • Test events require components, facilities, test equipment, etc.	• Model views answer questions for humans • Model views require model components, visualization capabilities, tool infrastructure, etc.

Descriptive models are like a system—that can (and should) be architected

Noguchi, R, Martin, JN, and Wheaton, M 2020, (MBSE)^2: Using MBSE to architect and implement the MBSE System, INCOSE International Symposium.

Foundations – Systems and software architecting

- These descriptive models constitute a unique kind of "system"
  - A system expressed in a form with some similarities to (and key differences from) software
  - A system intended to replace documents as a repository of knowledge
  - A system requiring a broader ecosystem of components to create, sustain, and fully utilize it

- Like any system or software product of non-trivial complexity, it will usually benefit from being consciously architected

- Architecture is:: "the fundamental concepts or properties of an entity in its environment and governing principles for the realization and evolution of this entity and its related life cycle processes" (ISO 42020:2021)
  - But the concepts or properties that are "fundamental" differ between systems, software, and models

What can we borrow from the disciplines of system and software architecting—and what do we need to adapt?

Foundations – Systems and software architecting

Systems Architecting

- System architecture is about:
  - The "*fundamental conception of a system*" that "*sets out what the parts of the system are, what they do, and how they fit and work together*." (Sillitto, 2014)
  - "an abstract description of the entities of a system and the relationship between those entities" (Crawley, 2016)
  - "the structure—in terms of components, connections, and constraints—of a product, process, or element" (Maier, 2009)

- System architecture can be characterized as a set of **decisions**
  - Decisions that are **foundational** to the concept of a system
  - Decisions that **distinguish** a given concept of the system from alternative concepts
  - Decisions that are **drivers** of cost, schedule, usability, evolutionary flexibility, and other considerations
  - Decisions that drive the growth or control of **technical debt**

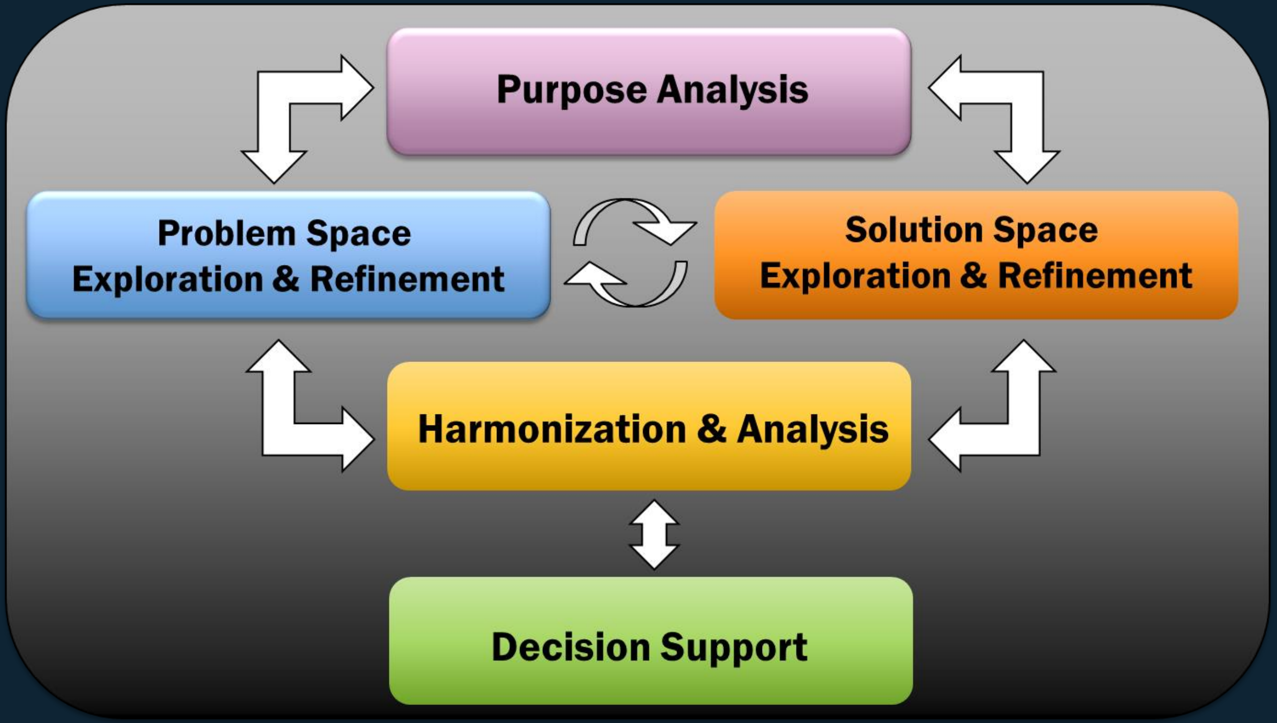
The role of the system architect is to make these decisions

Crawley, E, Cameron, B, and Selva, D 2016. *System Architecture: Strategy and Product Development for Complex Systems*, Pearson.
Maier, M and Rechtin, E 2009 *The Art of System Architecting*, 3rd *edition*, CRC Press.
Sillitto, H 2014, *Architecting Systems: Concepts, Principles, and Practice*, College Publications.

Foundations – Systems and software architecting

A Generalized Systems Architecting Process

- **Purpose Analysis**: Identify key **stakeholders** and **objectives** and establish the **scope** of the system's reason for being
- **Problem Space Exploration & Refinement:** Identify and refine **needs** and **constraints** and establish a value model for assessing alternative solution concepts
- **Solution Space Exploration & Refinement:** Develop candidate solution concepts
- **Harmonization & Analysis:** Identify feasible, compatible, and desirable subsets of the Problem Space and Solution Space by assessing alternative combinations against the value model
- **Decision Support:** Decide on the best courses of action; identify and prioritize the work needed to realize the concept


Purpose Analysis
Problem Space Exploration & Refinement
Solution Space Exploration & Refinement
Harmonization & Analysis
Decision Support

Systems architecting processes balance problem space and solution space analysis to inform architectural decisions

Noguchi, R, Martin, JN, and Wheaton, M 2020, (MBSE)^2: Using MBSE to architect and implement the MBSE System, INCOSE International Symposium.

Foundations – Systems and software architecting

Software Architecting

- Software architecture is about:
  - "The set of principal design decisions made about a system… the blueprint for a software systems' construction and evolution" (Taylor 2010)
  - "The set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both" (Bass 2012)
  - "The structure of the system… combined with the architecture characteristics ('-ilities') the system must support, architecture decisions, and finally design principles" (Richards 2020)
- Architecturally-significant requirements and key software quality requirements are among the most important drivers for quality software, particularly over the long term
  - e.g. performance, security, usability, robustness, etc.
- As with system architecture, software architecture can also be characterized as a set of decisions
  - Decisions that are **foundational** to the concept of a software product
  - Decisions that **distinguish** a given concept of the software from alternative concepts
  - Decisions that are **drivers** of cost, schedule, usability, evolutionary flexibility, and other considerations
  - Decisions that drive the growth or control of **technical debt**

The role of the software architect is to make these decisions

Bass, L, Clements, P, and Kazman, R 2012, Software Architecture in Practice, 3rd ed, Addison-Wesley.
Richards, M and Ford, N 2020, Fundamentals of Software Architecture, O'Reilly.
Taylor, R, Medvidovic, N, and Dashofy, E 2010, Software Architecture: Foundations, Theory, and Practice, Wiley.

Foundations – Systems and software architecting

Technical debt and its relation to architecting

- Technical debt is a concept widely used in the software community
  - The cost of required rework resulting from less-than-ideal architectural or design choices relative to the evolving needs and environment (Cunningham, 1992)
  - The concept is really more broadly applicable to domains well beyond software

- Technical debt is created whenever one chooses not to spend the time to “do it right”
  - Often time, budget, or resource constraints limit what can be accomplished
  - Often it isn’t clear what is “right”
  - Often what is “right” changes over time—what is “right” today may not be tomorrow

- In modern agile software development, the general principle most practitioners follow is to do “just enough” architecting and design to achieve near-term objectives
  - It is assumed that deferring work that doesn’t contribute to near-term goals is better than delaying feedback
  - It is assumed that the risk of rework is less than the risk of delayed feedback
  - It is unclear that these assumptions are as true for descriptive models as for software

The calculus of technical debt may not be the same for descriptive models

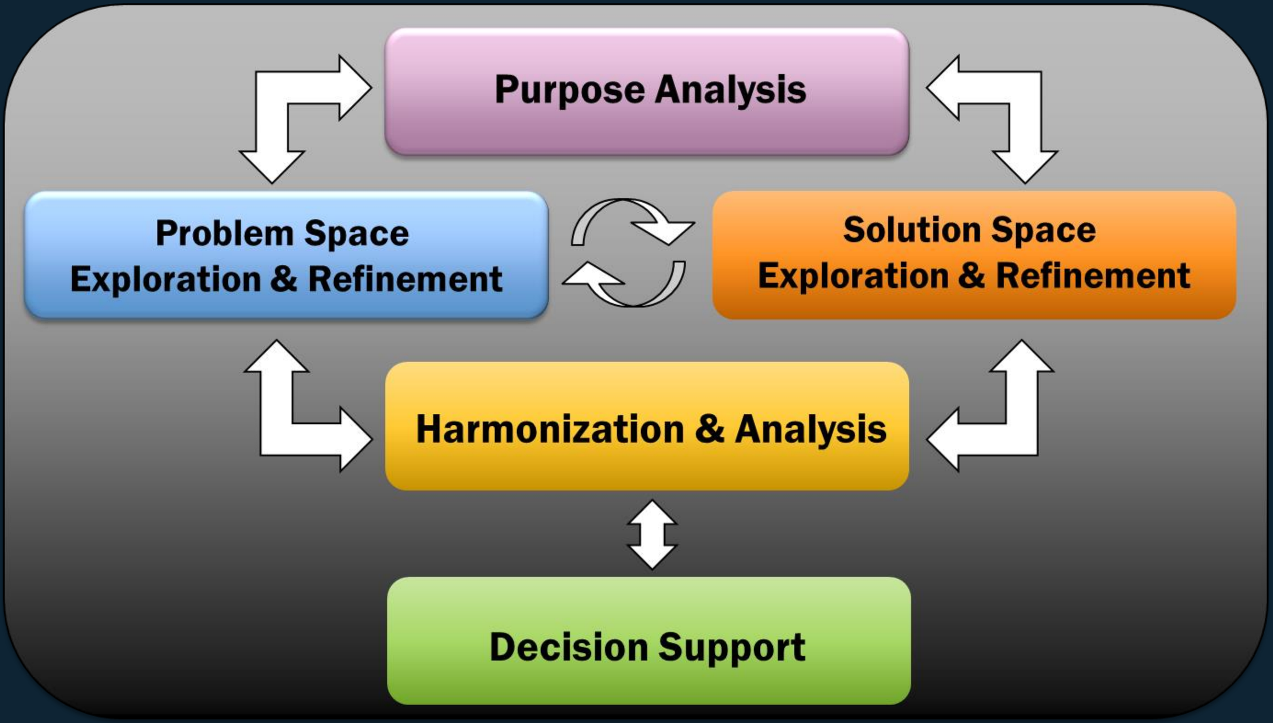
Cunningham, W 1992, The Wycash portfolio management system, in Addendum to the Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), ACM Press.

The discipline of (descriptive) model architecting

Applying systems architecting to models

- As with systems architecting, the role of the model architect is to make **architectural decisions** about models

- **Purpose Analysis**: Identify key stakeholders and objectives and establish the **model's** reason for being
- **Problem Space Exploration & Refinement:** Identify and refine the **questions** the models should answer and establish the value proposition for the models
- **Solution Space Exploration & Refinement:** Identify the **model content and form** to answer the questions
- **Harmonization & Analysis:** Identify **model views** that are needed to answer the questions and the model content and form needed to create those views
- **Decision Support:** Decide on the best courses of action; identify and prioritize the work needed to realize the concept and incrementally demonstrate value


Purpose Analysis
Problem Space Exploration & Refinement
Solution Space Exploration & Refinement
Harmonization & Analysis
Decision Support

Systems architecting processes can be adapted to architect models

Noguchi, R, Martin, JN, and Wheaton, M 2020, (MBSE)^2: Using MBSE to architect and implement the MBSE System, INCOSE International Symposium.

The discipline of (descriptive) model architecting

Applying software architecting to models

- As with software architecting, the role of the model architect is to make **architectural decisions** about models—particularly those that focus on **model form**
  - Software architecture practice largely focuses on relevant and significant structural relationships
    - "a structure is architectural if it supports reasoning about the system and the system's properties…. These include properties such as the functionality achieved by the system…. Architecture is concerned with the public side of this division; private details of elements—details having to do solely with internal implementation—are not architectural." (Bass, 2012)
    - "… architectural characteristics, the important aspects of the system independent of the problem domain … meets three criteria: Specifies a nondomain design consideration; Influences some structural aspect of the design; Is critical or important to application success." (Richards, 2020)

- Source code level implementation details are generally not considered to be architecturally significant
  - Since the software is as it does—the functionality is provided by the compiled object code, not the source code

- However, the functionality of descriptive models is largely exposed by its "source code"
  - What are "private details" in software are publicly exposed in descriptive models

Model architecting extends deeper into the implementation than does software architecting

Bass, L, Clements, P, and Kazman, R 2012, Software Architecture in Practice, 3rd ed, Addison-Wesley.
Richards, M and Ford, N 2020, Fundamentals of Software Architecture, O'Reilly.

The discipline of (descriptive) model architecting

Applying data architecting to models

- An organization's data can also be described with a similar architectural concept
  - How the data is organized, managed, collected, stored, transformed, distributed, curated, consumed, etc.
  - Database schemas, data formats, data structures, ontologies, metadata, etc.
  - Data can be organized in different architectural styles
    - e.g., data fabric vs. data mesh
  - Data can be modeled at multiple levels
    - e.g., Conceptual, logical, and physical data models

- Data architecture can be characterized as a set of **decisions** about the data
  - Decisions that are **foundational** to the concept of an organization's data
  - Decisions that **distinguish** a given concept of the data from alternative concepts
  - Decisions that are **drivers** of cost, schedule, usability, evolutionary flexibility, and other considerations
  - Decisions that drive the growth or control of **technical debt**

- The model architect needs to make decisions about the models that are then encoded in the model's data

The role of the model architect is to make key decisions about the model data

The discipline of (descriptive) model architecting

Applying document architecting to models

- A document could be considered to have an architecture of sorts
- Some of the foundational decisions that are drivers in the implementation of documents include:
  - What **content** will the document contain—and what will be intentionally omitted?
  - What are the key messages or **objectives** of the document?
  - How is the document **structured** and organized?
  - How are portions of the document cross-correlated or **cross-referenced**?
  - What **graphics** will be displayed to supplement the text?
- Descriptive models are largely intended to replace documents as the authoritative source of information for systems engineers and other stakeholders
  - What questions are the documents intended to answer?
  - Which of those questions do we intend the models to answer?
  - What documents do we need to generate from the models?
- The model architect needs to make decisions about the models like the documents the models replace

The role of the model architect is to make key decisions about the user experience of the models

Model architecture patterns and heuristics

Patterns

- Patterns have been used informally to inform architecture and design for millennia
- In the past quarter-century they have achieved more widespread recognition
  - Seminal work on pattern thinking and pattern language in civil architecture (Alexander, 1977)
  - Object oriented software design patterns (Gamma, 1995)
  - Data modeling patterns (Hay, 1995)
  - Higher level software architecture patterns (Fowler ,1997; Buschmann, 2007; Richards, 2015)
  - Enterprise architecture patterns (Perroud, 2013)

- Design patterns are intended to be well-tested solutions to well-defined, widely-observed design problems
  - Clearly identifying the **problem**
  - Illustrating the recommended **solution**
  - Identifying the **anti-pattern(s)**
  - Identifying **alternatives** and their **tradeoffs**
  - Identifying **constraints** and **contraindications**
  - Providing a memorable **name** for the pattern

Patterns are an essential tool in the architect's toolbox

Alexander, C, Ishikawa S, Silverstein, M, Jacobson, M, Fiksdahl-King, I, and Angel, S 1977, A Pattern Language: Towns, Buildings, Construction, Oxford University Press.
Buschmann, F, Henney, K, and Schmidt, D 2007, Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, Wiley.
Fowler, M 1997. Analysis Patterns: Reusable Object Models, Addison-Wesley.
Gamma, E, Helm, R, Johnson, R, and Vlissides, J 1995. Design Patterns: Elements of Reusable Object-Oriented Software,Addison-Wesley.
Hay, D 1995, Data Model Patterns, Addison-Wesley.
Perroud, T and Inversini, R 2013, Enterprise Architecture Patterns: Practical Solutions for Recurring IT-Architecture Problems, Springer.
Richards, M 2015, Software Architecture Patterns: Understanding Common Architecture Patterns and When to Use Them, O'Reilly Press

Model architecture patterns and heuristics

Heuristics

- Closely related to patterns are heuristics
- In system architecting, heuristics represent abstractions of experience that help to guide the architect in the right directions to improve their chances of developing a good architecture
  - A compendium of these heuristics for system architecting was described by Maier and Rechtin (Maier, 2009)

- Heuristics are also commonly seen in the software domain
  - Generally the lines between these heuristics and patterns is blurrier than is the case in systems architecting
  - Heuristics for software design and implementation were documented shortly after the Gamma, et al. book popularized design patterns (Riel, 1996)
  - Well-known heuristics for good object oriented design are referred to by the acronym SOLID (Martin, 2006)
    - Single Responsibility Principle
    - Open/Closed Principle
    - Liskov Substitution Principle
    - Interface Segregation Principle
    - Dependency Inversion Principle

Heuristics are an essential tool in the architect's toolbox

Maier, M and Rechtin, E 2009 The Art of System Architecting, 3rd edition, CRC Press.
Martin, R and Martin, M 2006, Agile Principles, Patterns, and Practices in C#, Pearson.
Riel, A 1996, Object-Oriented Design Heuristics, Addison-Wesley.

Model architecture patterns and heuristics

Model federation patterns and heuristics

- As with systems or software, a large monolithic model can be unwieldy to develop, use, and evolve
  - Large systems or software products are **partitioned** into an integrated set of smaller components to facilitate:
    - Distributed development
    - A modular open systems approach (MOSA)
    - Controlled reuse
- Models can be **federated** to enable the integration of the knowledge they contain
  - Federated models retain their identity and serve their local intended purpose while also contributing to broader purposes
    - Models in a federation are like constituent systems in a **system of systems**
  - Partitioning the total model scope into a set of smaller models and establishing formal relationships between those models is one of the key tasks of the model architect, who must seek to avoid:
    - Interdependencies (cyclical dependencies) between models
    - Overpartitioning or underpartitioning
- Defining the preferred modeling approach for connecting models in a federation is critical
  - There are multiple approaches for implementing model federation, each with their own tradeoffs

The model architect has critical decisions to make about model federation architecture and implementation

Martin, J and Brookshier, D, 2023, Linking UAF and SysML Models: Achieving Alignment Between Enterprise and System Architectures, INCOSE International Symposium.

Swickline, C, 2023, A Methodology for Model Federation Applied Across Defense Systems Development Programs, INCOSE International Symposium.

Vinarcik, M 2022, A Mars Octet: lessons learned from federating eight student models in a SysML class, AIAA SciTech 2022 Forum.

Model architecture patterns and heuristics

Intra-model modeling patterns and heuristics

- Within the context of an individual model, many of the model architecture decisions to be made involve the organizational structure of the model and the domain-specific language used to express the model
  - When models will need to be federated, these choices are often constrained to facilitate model interoperability
  - Some standard architecture frameworks—like the Unified Architecture Framework (UAF)—have already implemented some model federation principles through the organization of standard views into domains or viewpoints

- The SOLID object-oriented software design principles can be adapted to the modeling domain

SOLID Principle	Interpretation for OO Software	Interpretation for OO Models
Single Responsibility Principle	A class should be minimal in scope to reduce its need for change	A model element type definition should be minimal in scope, representing a single coherent concept rather than a combination of multiple concepts
Open/Closed Principle	A class should be open for extension but closed for modification	A model element type definition should be designed to be specialized without having to be modified
Liskov Substitution Principle	A subclass should always be substitutable for its base class	Any specialization of a base classifier should be a valid substitute for that base classifier
Interface Segregation Principle	A class should not implement an interface it doesn't use	A model element type definition should avoid defining features—particularly behaviors—that are not required by all of its specializations
Dependency Inversion Principle	Concrete modules should depend on abstractions rather than on other concrete modules	Model dependencies should be directed toward increased abstractions rather than the other way around

Capturing these patterns and heuristics and coalescing to a common vocabulary to refer to them is a practice that the MBSE community should pursue to accelerate its maturation as a discipline

Model architecture patterns and heuristics

“Atomic” modeling patterns and heuristics

- Many of the modeling patterns the model architect needs to consider are at a much lower level of abstraction than is typical of software architecture
  - Selecting the modeling constructs to use to express
  - In software, choosing between programming language constructs (e.g., while vs. for) is not considered architectural
- However, these sorts of decisions are much more impactful for models since they can have significant consequences for the interpretation and use of the models
  - Model users interact directly with the models—the equivalent of software source code
  - The value they extract from the model is largely obtained through direct inspection of model views in which the model’s construction is fully exposed
- Furthermore, descriptive models are often much more difficult to refactor
  - Refactoring in software is updating source code, without changing its externally visible behavior, to improves its maintainability or to address performance problems (Fowler, 1999)
  - Refactoring in software can usually be done piecemeal and incrementally
  - Refactoring in models is often much more difficult, since the changes are generally much more pervasive, invasive, and interdependent—and the model is left in an inconsistent state until the refactoring is complete

The model architect often has to make decisions that may seem like “implementation details” but are often actually architecturally significant

Fowler, M 1999. Refactoring: Improving the Design of Existing Code, Addison-Wesley.

Model architecture patterns and heuristics

“Atomic” modeling patterns and heuristics—examples

- Choosing which relationship type(s) to use to connect two model elements to represent a given assertion
  - e.g., to assert an equivalency relationship between a model element representing System X in a UAFML model and a model element representing that same System X in a separate SysML model, what relationship should be used?
    - Allocation? Reference association? Generalization?
    - Some specialization of one of those relationships?
- Choosing a mechanism for implementing an attribute of a concept
  - e.g., if a model element represents a person that has an organizational role, how should that role be modeled?
    - Is the role the value of a value property owned by the person?
      - If so, should it be represented by a String or an enumeration?
    - Or is the role a separate model element that should have a relationship to the person?
      - If so, what kind of relationship should this be, and which model element should own that relationship?
    - Should the role be a globally-scoped element—e.g., a model-equivalent of the Singleton OO design pattern?
- Unfortunately, these decisions are often made implicitly
  - Without consideration of alternative approaches
  - Without a deep understanding of the implications and tradeoffs
  - Without any formal documentation of the design decision made and its rationale
  - As a result, model construction is still quite a bit less mature in practice and discipline than software development

Making the wrong choice can result in long, tedious, error-prone refactoring in your future!

Role and competencies of the Model Architect

Role of the model architect

- The role of the model architect is to make the big architectural decisions that drive the model's value in both the near term and the long term
  - These decisions need to be made early enough to guide the modeling effort
    - And decisions continue to be necessary throughout the model's life
  - Typically some long-term benefits must be sacrificed to achieve near-term benefits
    - Often this results in the accumulation of technical debt
    - The model architect needs to decide what technical debt is worth accepting

- A good model architect must be able to flex to address the big picture questions as well as the details
  - They must be well-versed in the constraints and the finer points of the modeling languages and tools
  - They should understand alternative modeling approaches and their tradeoffs
  - They should be able to work with stakeholders in their language and with modeling and tool SMEs in theirs

To make the right model architectural choices, the model architect needs to balance strengths in working the problem space and the solution space

Role and competencies of the Model Architect

A cautionary tale—where experience may be dangerous

- Since descriptive models are typically used to support MBSE, prospective model architects often have experience as a systems engineer or software developer
  - This doesn't necessarily translate into an understanding of architecting, which is a subset of the skillset (and mindset)
- A model architect with experience in software architecting may encounter paradoxes that challenge the orthodoxy of their background
  - Experienced software developers often have a bias against multiple inheritance
    - While multiple inheritance is often unnecessary—and easily worked-around—in software coding, it is much more commonly needed in descriptive modeling
  - Experienced software developers may subvert natural expectations of inheritance due to their experience with paradoxical situations in OO programming
    - The classic "square-rectangle" or "circle-eclipse" problem in OO reveals that inheritance in software is primarily about the behavior of objects rather than about their intrinsic nature
    - Software is as software does, but modeling must respect the intrinsic qualities of the subjects being modeled
- Growing a cadre model architects requires exposure to best practices, lessons learned, and experiences

The model architect is a bit of a unique beast—and is essential to the long-term success of MBSE

Key Takeaways

- Descriptive models have a combination of characteristics of documents, data, and software
- Descriptive models are like a system… constructed somewhat like software… implemented as data… and intended to replace documents
- Applying some of the key principles and methods of system and software architecting is essential to achieve success in MBSE
- Architecturally significant decisions for models can be similar to those for systems and software… but generally dive deeper into the implementation than is the case for either
- Modeling patterns and heuristics are an essential component of a mature architecting discipline
- Experience in system or software architecture is helpful… but there are potential pitfalls to be aware of

A good model architect is priceless!

Questions?

Ryan Noguchi
Principal Engineer
The Aerospace Corporation
ryan.a.noguchi@aero.org

33rd Annual INCOSE international symposium

hybrid event

Honolulu HI USA

www.incose.org/symp2023
#INCOSEIS