



NAVAL  
POSTGRADUATE  
SCHOOL

# **Behavioral Modeling of Software System Architectures and Verification & Validation**

Monica Farah-Stapleton  
Professor Mikhail Auguston  
Professor Kristin Giammarco

Monterey, California

[WWW.NPS.EDU](http://WWW.NPS.EDU)

24 September 2015





- **V&V Summit Key Objectives**

- Foster a V&V best practices and corporate V&V philosophy
- Explore new and practical ways to apply V&V that better support acquisitions and decision making
- Promote V&V disciplines and culture
- Highlight “real world” ways to incorporate V&V into organizational operations

- **Today’s Presentation**

- Common Mental Model: Challenge and Response
- Behavioral Modeling Using Monterey Phoenix (MP)
- Inform Resourcing Decisions
- Discipline and Organizational Implications
- Some Thoughts



- Information Technology (IT) systems are large, challenged by the rate of change of commercial IT, and represent a significant investment in time and resources.
- The introduction of a new system or capability may result in unintended or unexpected system and environment behaviors that have operational and financial impacts. These impacts are often assessed after the fact.
- Precise behavioral modeling offers a way to assess architectural design decisions and their impacts prior to, during, and after implementation and deployment.
- Precise architectural descriptions of system/environment and resourcing decisions are often minimally related.



- **Verification:** *Building the system right*
  - Ensures that selected work products, product components, and products meet specified requirements and standards
  - Inherently an incremental process since it occurs throughout the development of work products and products
  - Beginning with initial concepts, progressing through subsequent changes, and continuing throughout the lifecycle
- **Validation:** *Building the right system*
  - Demonstrates whether a product will fulfill its specified purpose when placed in any aspect of its intended environment such as operation, training, manufacturing, maintenance, or support services
  - Methods employed to accomplish validation can be applied to work products as well as to the product
  - Work products are selected on the basis of being the best predictors of how well the product and product component will satisfy user needs and the level of risk they present to the program
  - Validation is performed early and incrementally throughout the product lifecycle, often requiring rigorous analysis to ensure the right product is being procured



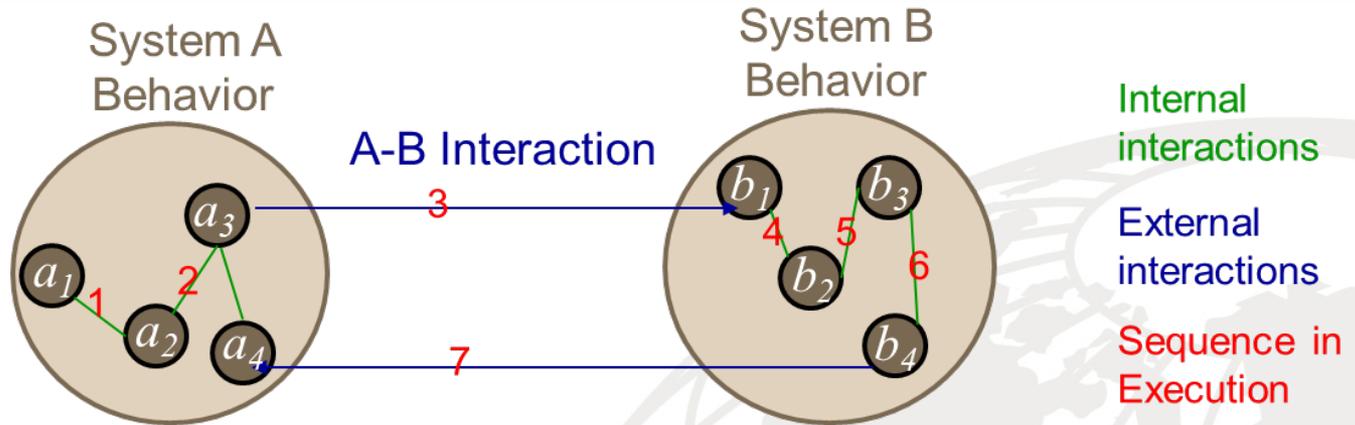
System's  
Architecture



Stakeholder Examples	Typical Questions or Groups of Questions
Customer	Are user, technical, cost, and management expectations being met?
Users	Does this system do what was expected? Does it fulfill prioritized requirements?
Engineers / Designers	What implementation option(s) should be considered to meet performance expectations? What are environment interactions and constraints for each option?
Testers	What are optimal instrumentation points? What statistics should be gathered? What is the correct level of abstraction?
Cost Analysts	What is the cost of the system from requirements elicitation thru software evolution?

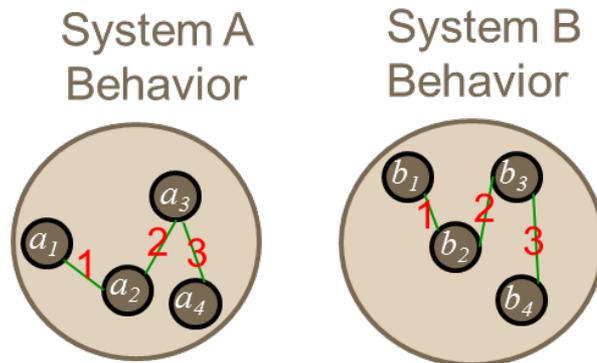
# What Does Separation of System Interaction from System Behavior Mean?

**From...**

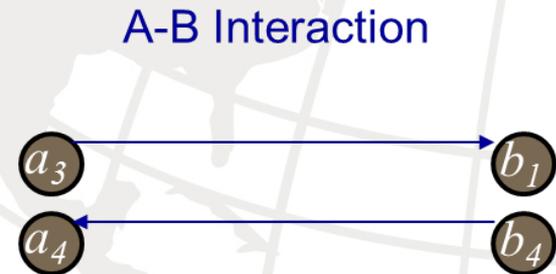


↑ Sequencing *external* interactions in the same model as *internal* interactions

**To...**



↓ Specifying behavior of each system *separately* from interactions among those systems, and then compute possible sequences





- An approach to formal software system architecture and business process (workflow) specification based on *behavior models*
- A view on the architecture as a high level description of possible system behaviors, emphasizing the behavior of subsystems and *interactions* between subsystems
- The emphasis on specifying the interaction between the system and its *environment*
- The *behavior composition operations* support architecture reuse and refinement towards design and implementation models
- Separates specification of system *interaction* from system *behavior* -- separation of concerns
- *Executable architecture* models provide for system architecture testing and verification with tools



- An *executable* system architecture model -- Monterey Phoenix scenario generator can produce event traces with several hundred or thousands of events
- An event trace *visualization framework* that enables human analysts to focus on the behavior of the system and provides *multiple views* for different stakeholders
- Mechanisms to run *queries* on the automatically generated event traces, and a language for event trace analysis (*assertion checking*)

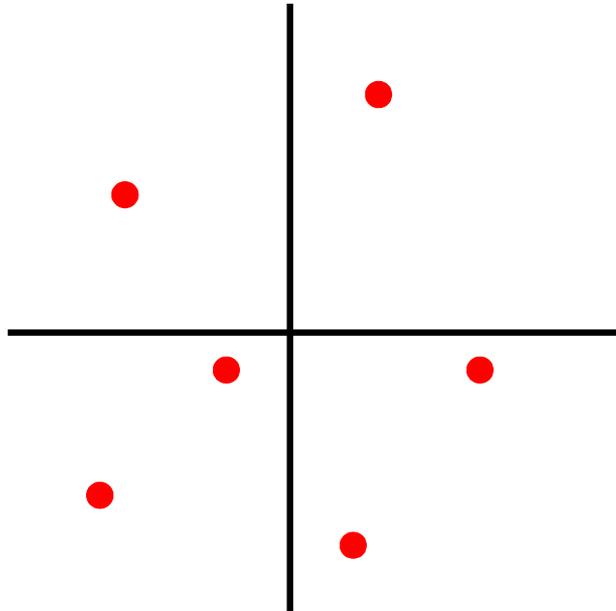
*Architecture of a system is considered in the context of the environment in which it operates, including business processes*



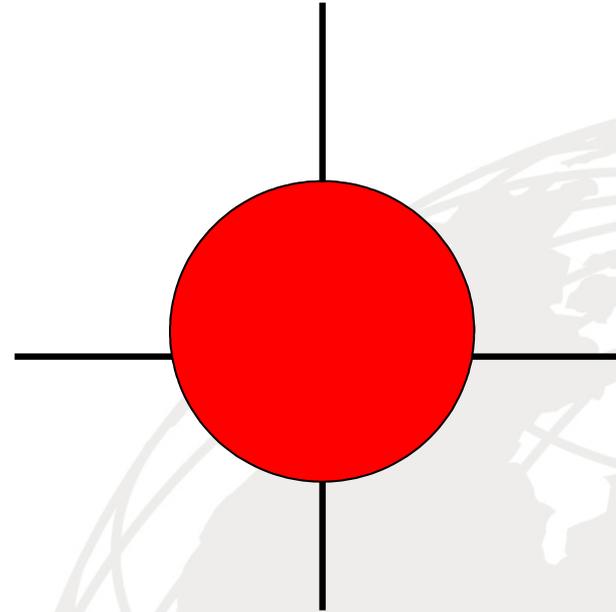
# Architecture Verification & Validation

- Means to write *assertions* about the system behavior and tools to verify those assertions.
- Integration of the architecture models with *environment models* for verifying system's behavior on typical scenarios (Use Cases).
- *Event attributes*, like timing, can be used for non-functional requirements (like performance estimates) V/V and queries (like critical path estimates in PERT diagrams).
- Assigning *probabilities* to certain events makes it possible to obtain statistical estimates for system behaviors.
- Scenario inspection in MP can be *automated* by assertion checking tools
- Interactions of subsystems and environment can be used for detecting unexpected and *emergent behaviors* in System of Systems
- *Different views* can be automatically extracted and visualized for different stakeholder needs -- it is much easier for different stakeholders *to understand and verify stand-alone scenarios (Use Cases)*

# Model Verification Within Limited Scope

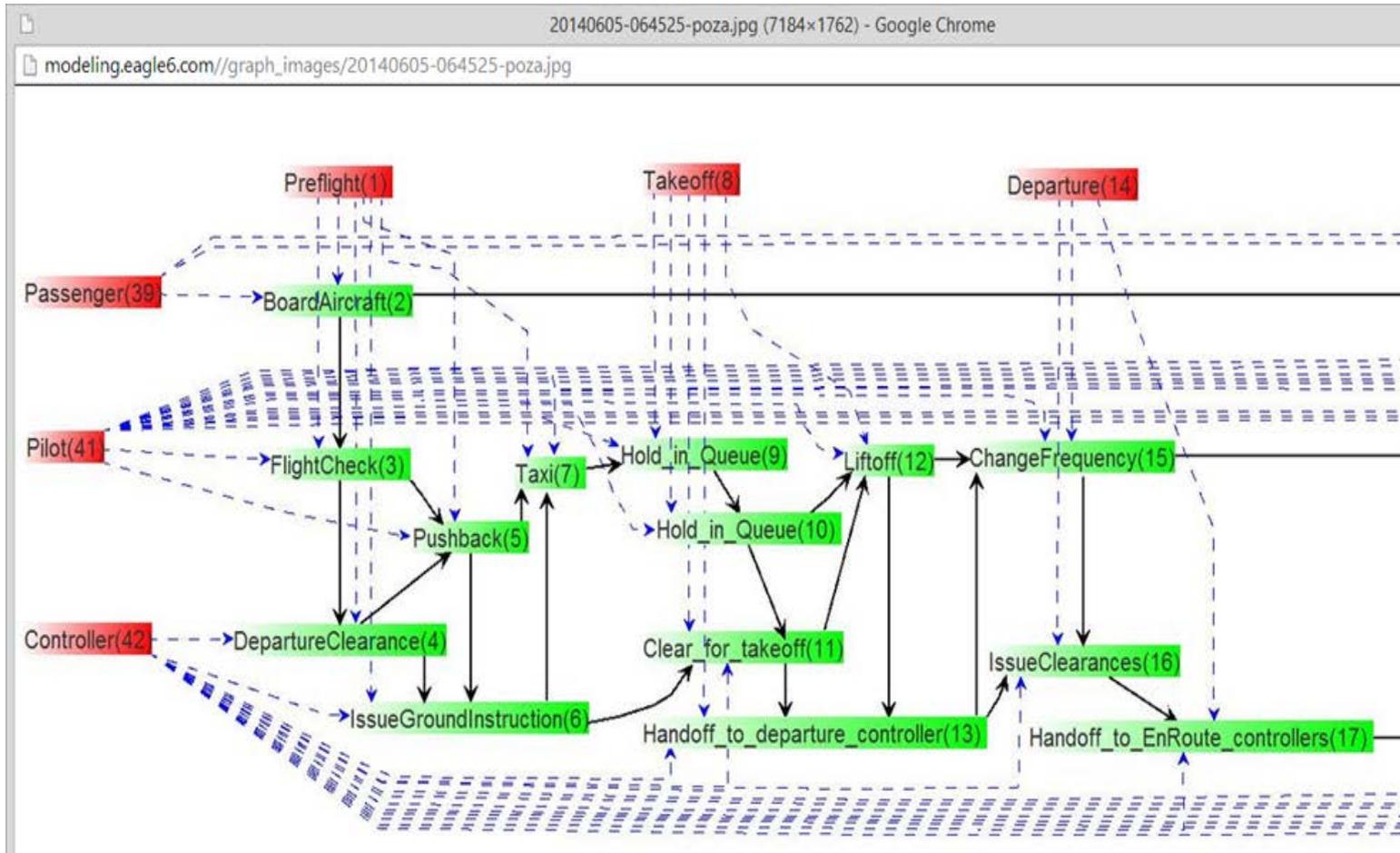


Testing:  
A few cases of arbitrary size

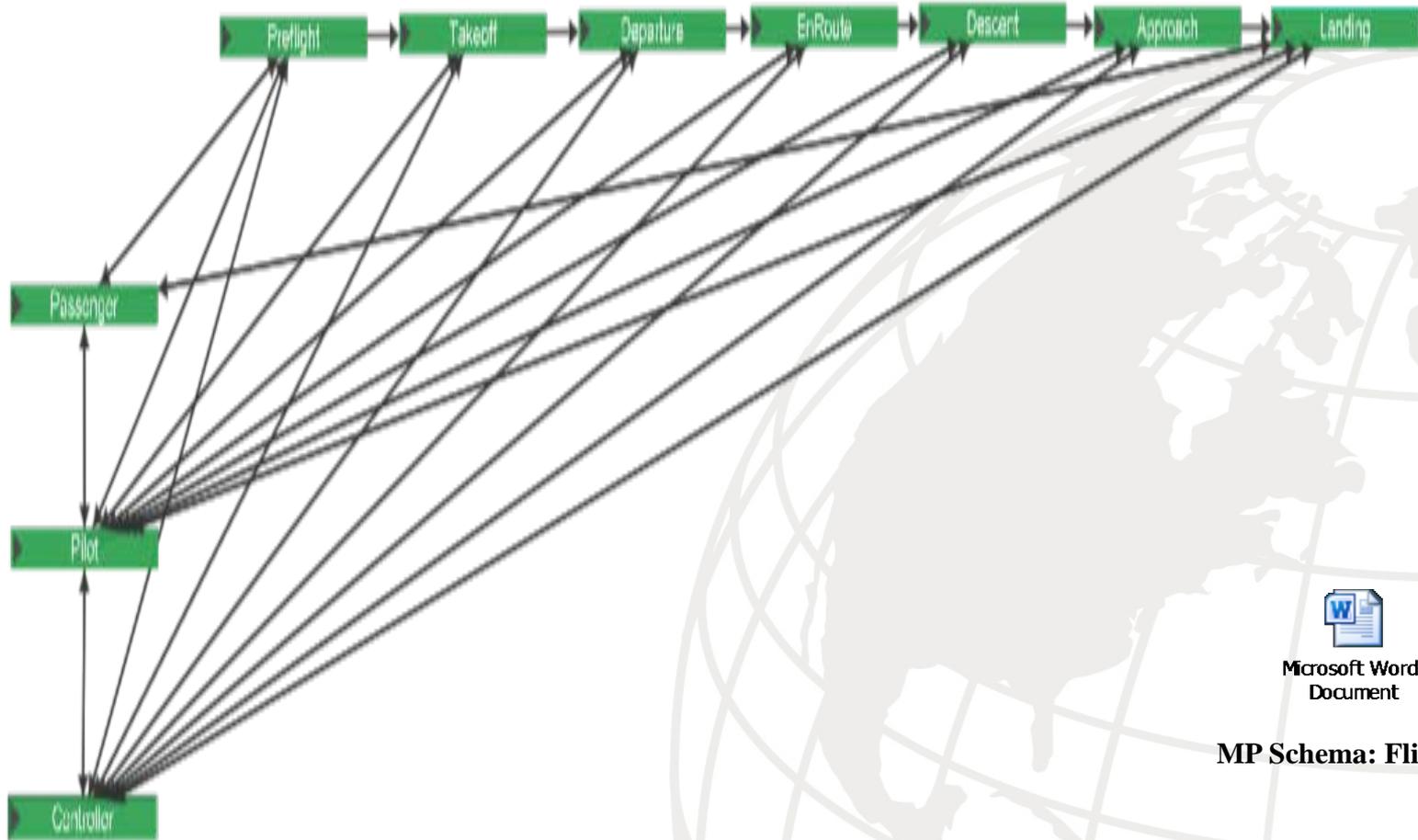


Scope-complete:  
All cases within a small  
bound

- **Exhaustive search** through all possible scenarios (up to the scope limit)
- **Small Scope Hypothesis:** Most flaws in models could be demonstrated on small counterexamples



Sample Portion (first three phases) of one of 32 possible scenarios generated by Eagle 6 (scope 3).  
Eagle 6 Prototype Tool: <http://eagle6modeling.riverainc.com/>

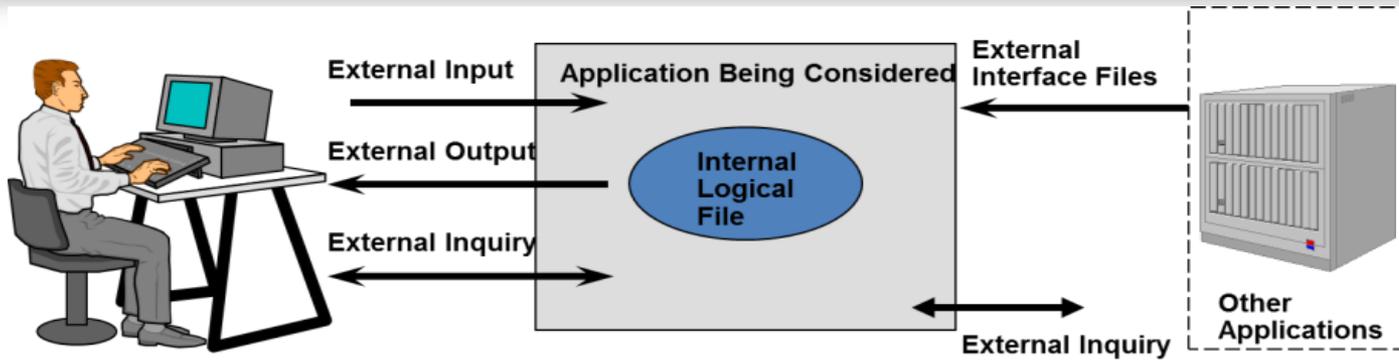


Public MP server with MP editor, trace generator, and trace graph visualization:

<http://firebird.nps.edu/>

# Function Point Analysis (FPA)

## Functionality From User's Perspective



### Function Point Terminology

- Describe interactions of a user, system and its environment.
- External Inputs (EI): Data that is entering a system.
- External Outputs (EO) and External Inquires (EQ): Data that is leaving the system.
- Internal Logical Files (ILF): Data that is processed and stored within the system.
- External Interface Files (EIF): Data that is maintained outside the system but is necessary to satisfy a particular process requirement.

### Function Point Analysis Practice

- Count Data and Transactional Function Types.
- Determine Unadjusted FP (UFP)
- Determine the Value Adjustment Factor – N/A
- Calculate final Adjusted FP Count – N/A

Function Points: Normalized metric used to evaluate software deliverables and to measure its size based on well-defined functional characteristics of the software system. Must be defined around components that can be identified in a well-written specification.

Source: International Function Point User Group, inspired by "Introduction to the International Function Point Users Group (IFPUG)", p.6, © Copyright 1999, International Function Point User Group 1999 and <http://www.ifpug.org/>

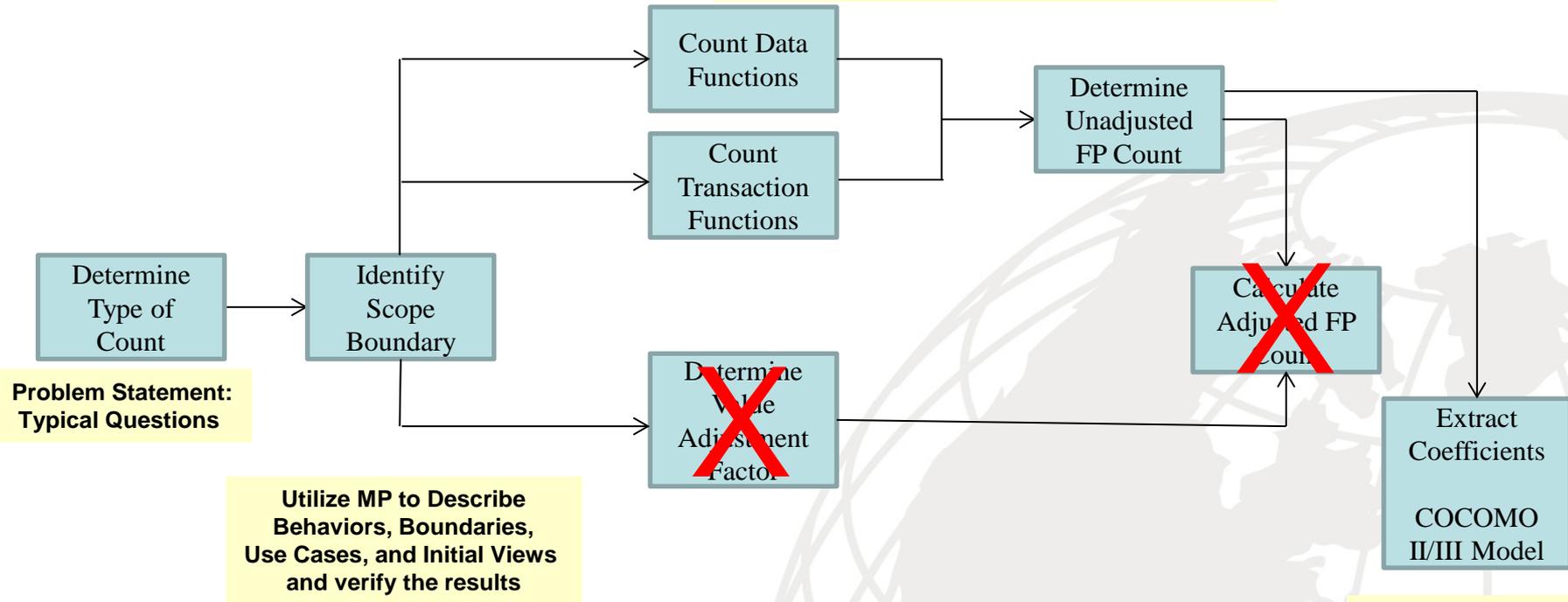


# Calculate UFP for Software of Interest

Correlate MP descriptions of behaviors with FPA descriptions of behaviors

- MP is in the event space – to use FPA the events have to be described in a way that relate them to data/transactional functions and FP types. So in the FPA world, words like add, change, delete, have to relate to high level pseudo code descriptions of the behavior of the ROOTS (actors) and their interactions via COORDINATE and SHARE ALL.
  - External Inputs (EI): Input data that is entering a system (application) under analysis.
  - External Outputs (EO) and External Inquires (EQ): Data that is leaving the system under analysis.
  - Internal Logical Files (ILF): Data that is processed and stored within the system under analysis.
  - External Interface Files (EIF): Data that is maintained outside the system under analysis but is necessary to satisfy a particular process requirement.
- Identify all functions: ILF, EIF, EI, EO, EQ
- Rate the complexity of ILF, EIF, EI, EO, EQ
- Assign the function point values and total

Apply Function Point Counting Methodology



- Assess Complexity and Scale thru Coefficients
- Assess Effort and \$ thru - COCOMO II/III Model - # FPs/Hour

- Step 1:** Identify the problem statement, i.e. what are the typical questions to be answered
- Step 2:** Describe the behaviors of the system and environment in natural language
- Step 3:** Unambiguously represent behaviors using MP, extract Use Cases/initial views from MP model
- Step 4:** Perform Architecture verification and validation, and assertion checking
- Step 5:** Un-ambiguously relate system and environment behaviors to Function Point behaviors
- Step 6:** Extract coefficients that inform complexity and scale
- Step 7:** Determine Unadjusted Function Point (UFP) count, insert into COCOMO II/III or identify # FPs/hour
- Step 8:** Assess assumptions of complexity as related to cost estimates
- Step 9:** Visualize results in views specific to stakeholders



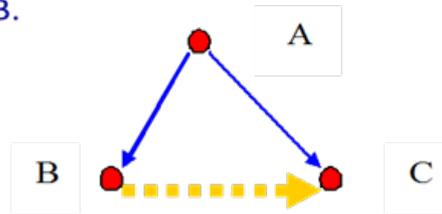
- V&V is not isolated to the technical domain but rather the socio-technical domain
  - Business process modeling can add efficiency to an organization
  - Applicable to SoS concepts, architectures, business practices, organizational dynamics
- Executable behavioral modeling of system and software architecture specifications, leveraging lightweight formal methods and pseudocode can assist in promoting a V&V culture
  - Precise and user-friendly tools
  - Enforce use of tools across the organization
- Resourcing decisions shouldn't be arbitrary
  - Cost is an attribute of an instance of an architecture
  - Effective and efficient are not mutually exclusive



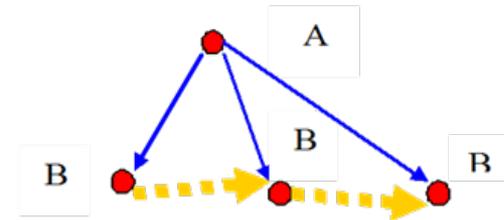
- **Event** - any detectable action in system's or environment's behavior
- **Event trace** - set of events with two basic partial ordering relations, **precedence** (PRECEDES) and **inclusion** (IN)
- **Event grammar** - specifies the structure of possible event traces



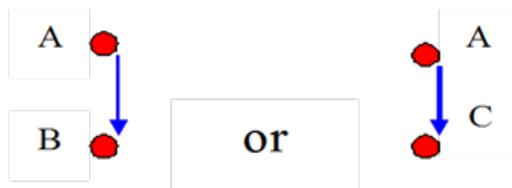
The rule  $A:: B C$ ; specifies the event trace sequence of zero or type B.



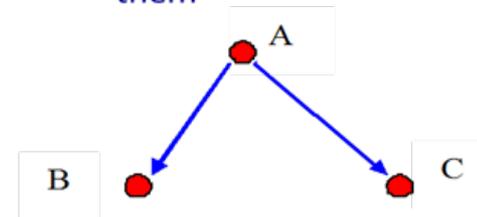
$A:: (* B *)$ ; means an ordered more events of the



$A:: (B | C)$ ; denotes alternative



$A:: \{ B, C \}$ ; denotes a set of events B and C without an ordering relation between them





- The architect needs *different views* for the various uses and users
- A system architecture description belongs at a *high level of abstraction*, ignoring many implementation details (e.g. algorithms and data structures)
- The architecture plays a role as the *bridge between requirements and implementation* of a system
- Errors *in early system design* are the most expensive to fix when detected later in the development lifecycle
- *Modeling* is an approach to the design and verification of system architecture
- The architect needs *different views* for the various uses and users
- One major concern in architecture design is the *behavior* of the system
- Architecture specification should be supportive of the *refinement* process
- Composition operations focus on *interactions* between the parts of the system
- Architecture of a system is considered in the context of the *environment* in which it operates, including *business processes*