**14th NDIA Systems Engineering Conference**

**24-27 October 2011**

***Tutorial 13122***

# Systems Architecting: Practices for Agile Development in the Systems Engineering Context

Tommer R. Ender, PhD
Senior Research Engineer
tommer.ender@gtri.gatech.edu
(404) 407-8639

Tom McDermott
Dir. of Research and Dep. Dir., GTRI
tom.mcdermott@gtri.gatech.edu
(404) 407-8240

Nicholas Bollweg
Research Engineer I
nicholas.bollweg@gtri.gatech.edu
(404) 407-7207

Georgia Tech Research Institute

Problem. Solved.

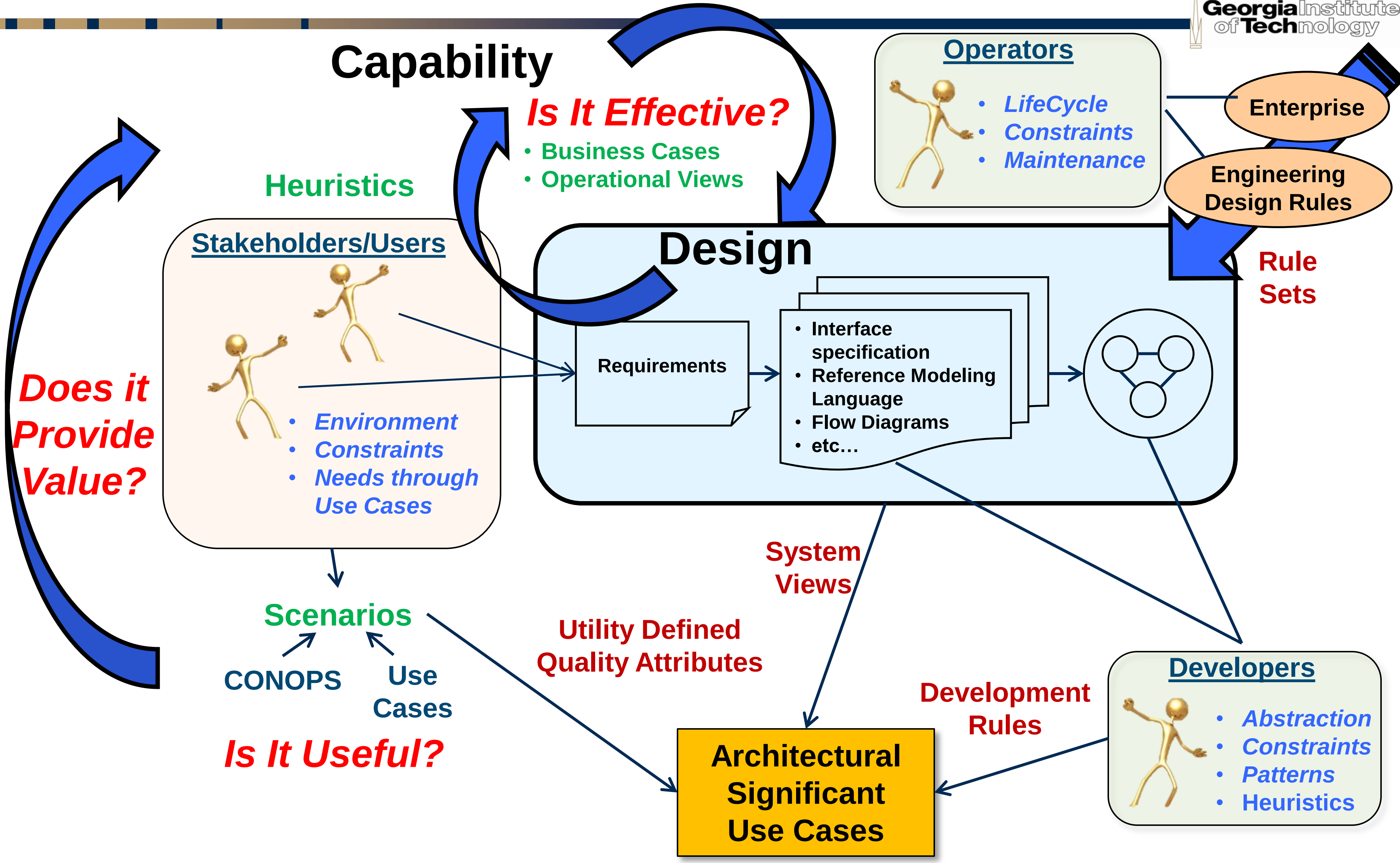
# Goals and Learning Objectives

- Introduce the student to **methods** and **practices** for systems architecting
- Apply **agile principles** and incremental development to architecting
- Learn **novel methods** for combining narrative, visual, and specification techniques for rapid and incremental architecture development
- Learn practical approaches to **facilitate the process** introduced in this tutorial

# Summary of Topics

- Fundamental **systems architecting**
- Incremental development of ill defined or evolving systems through **agile development**
- Evaluating architecture quality through **scenario based methods** is reviewed in the context of satisfying business drivers
- Practical **management methods** are introduced focusing on the leadership role of the systems architect on a development team

# Narrative Context



**Capability**

*Is It Effective?*
- Business Cases
- Operational Views

**Heuristics**

**Stakeholders/Users**
- *Environment*
- *Constraints*
- *Needs through Use Cases*

*Does it Provide Value?*

**Design**

Requirements

- Interface specification
- Reference Modeling Language
- Flow Diagrams
- etc…

**Operators**
- *LifeCycle*
- *Constraints*
- *Maintenance*

Enterprise

Engineering Design Rules

Rule Sets

Scenarios

CONOPS

Use Cases

*Is It Useful?*

Utility Defined Quality Attributes

System Views

Development Rules

**Architectural Significant Use Cases**

**Developers**
- *Abstraction*
- *Constraints*
- *Patterns*
- Heuristics

# Systems Architecting: Practices for Agile Development in the Systems Engineering Context

## Architecting Models

# Systems “Architecting” vs. “Engineering”

- Systems architecting differs from systems engineering in that it relies more on **heuristic** reasoning and less on use of analytics

- There are **qualitatively** different problem solving techniques required by high and low complexity levels
  - The lower levels would certainly benefit from purely analytical techniques, but those same techniques may be overwhelming at higher levels which may benefit more from heuristics derived from experience, or even abstraction
  - It is important to concentrate on only what is essential to solve the problem

***The system should be modeled at as a high a level as possible, then the level of abstraction should be reduced progressively as needed***

# Architecture Definitions

- **Architecture**: the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution
- **Architecting**: the activities of defining, documenting, maintaining, improving, and certifying proper implementation of an architecture
- **Architectural Description**: a collection of products to document an architecture

*Source: IEEE-1471-2000*

# Classical Architecting Methods

- **Science based**
  - Analytic, deductive, experiment based, easily certified, well understood, widely taught

- **“Art” or practice of architecting**
  - Nonanalytic, inductive, difficult to certify, less understood, seldom formally taught
  - Process of insights, vision, intuitions, judgment calls, subjective “taste”
  - Deals with immeasurables, sanity checks
  - Leads to “**unprecedented** systems”

# Insight

- The ability to structure a complex situation in a way that greatly increases understanding of it
- Guided by lessons learned from experience and observations
- Where systems architecting becomes more an art than a science

*Success comes from wisdom…*

*Wisdom comes from experience…*

*Experience comes from mistakes*

**Those mistakes and experience may come from one's predecessors**

***Insight = Heuristics***

# Heuristic Methods

- Based on prior experience and common sense (what is sensible in a given context)
- Collective experience stated in as simple and concise a manner as possible
- Provide practical and pragmatic guidance through intractable or “wicked” problems

# Heuristics

- A concise statement of situational insight, lesson learned, or design directive
  - *"All the really important mistakes are made the first day"*
  - *"When partitioning a system choose so that elements have high internal complexity and low external complexity (high cohesion and low coupling)"*
  - *"if the politics don't fly, the airplane never will"*
- Maier (2009) has compiled a list of "heuristics for systems level architecting" in an appendix
  - Multitasking
  - Scoping and planning
  - Modeling
  - etc…

Useful to review relevant and define applicable heuristics before undertaking a new effort…identify potential roadblocks!

# Complexity

- **Complex**: composed of interconnected or interwoven parts
- **System**: a set of different elements so connected or related as to perform a unique function not performed by the elements alone

- ***Is a system, by definition, complex?***
  - **Complexity**: the measure of the numbers of types of interrelationships among system elements
  - the more complex a system, the more difficult to design, build, and use

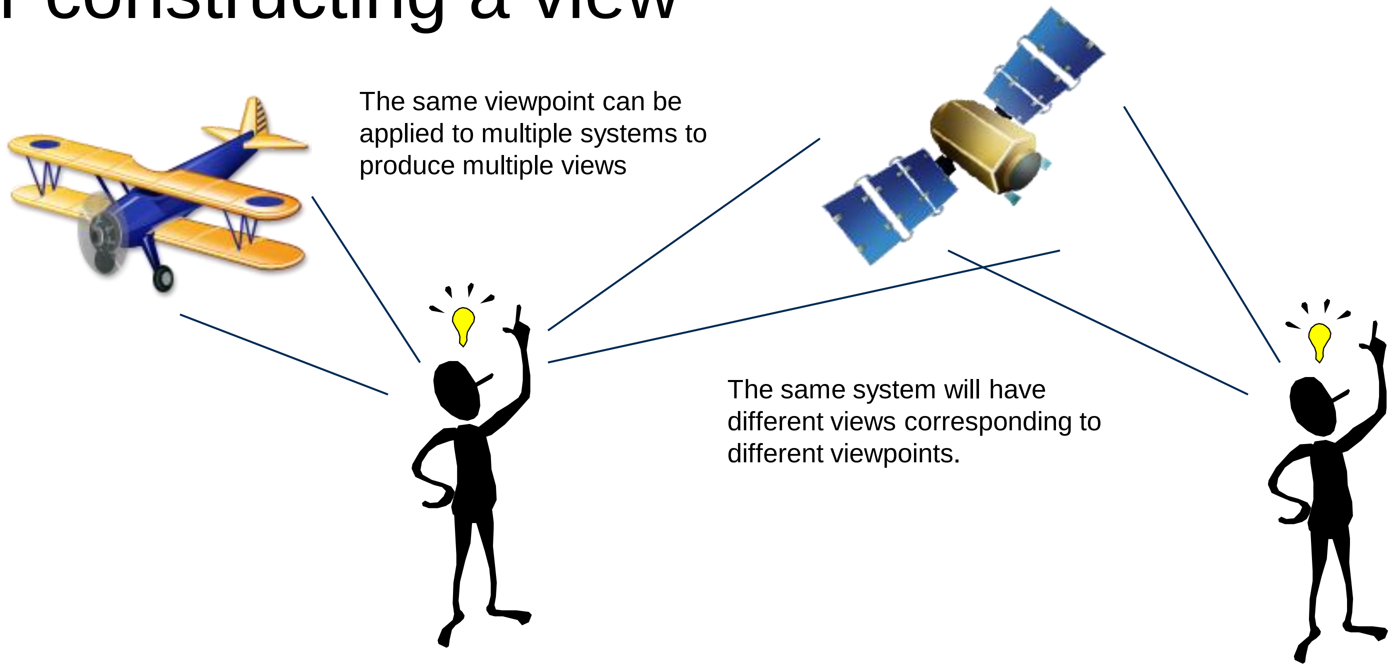
# Normative Requirements for Architecture Description

- The stakeholders identified must include users, acquirers, developers, and maintainers of the system
- The architectural description must define its viewpoints, with some specific elements required
- The system's architecture must be documented in a set of views in one-to-one correspondence with the selected viewpoints, and each view must be conformant to the requirements of its associated viewpoint
- The architecture description document must include any known interview inconsistencies and a rationale for the selection of the described architecture

*source: IEEE-1471-2000 ; Maier (2009)*

# Views and Viewpoints

- A **View** is a representation of a system from the perspective of related concerns or issues
- A **Viewpoint** is a template, pattern, or specification for constructing a view



The same viewpoint can be applied to multiple systems to produce multiple views

The same system will have different views corresponding to different viewpoints.

Viewpoint consists of:
✓ Concerns (of the Stakeholder)
✓ Methods

*terms: IEEE-1471-2000*
*Graphics adapted from: Maier (2009)*

# Views and Viewpoints

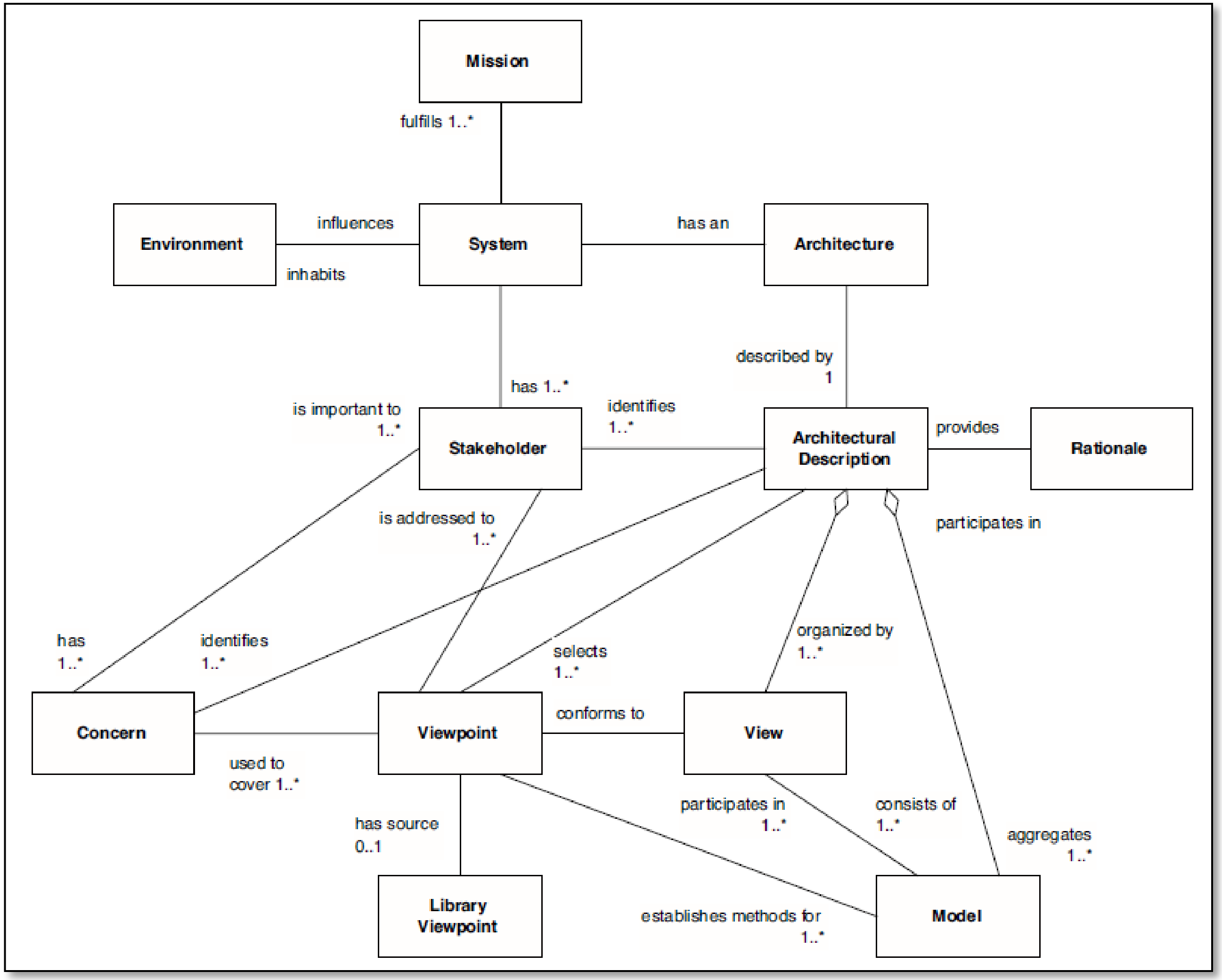
- A **view** is a collection of models that share the same concerns of a stakeholder
  - **Classical architecture**: shows physical properties of a building from a particular perspective (i.e. a floor plan)
  - **Systems architecting**: generalizes when physical property is not primary, but includes functionality (and others)
- A **viewpoint** is an abstraction of the view across many systems

# Views for Describing a System

- A view **describes a system w.r.t. a set of attributes** and/or concerns
- The views selected are **problem dependent** (i.e. variable), however….
- Should be **complete**: the complete set of views should cover all stakeholder concerns
- Should be **independent**: each view should capture different piece(s) of information
    - » ***Independent? Well, kind of….(more on this later)***

# IEEE-1471-2000:
## Conceptual Model of an Architectural Description

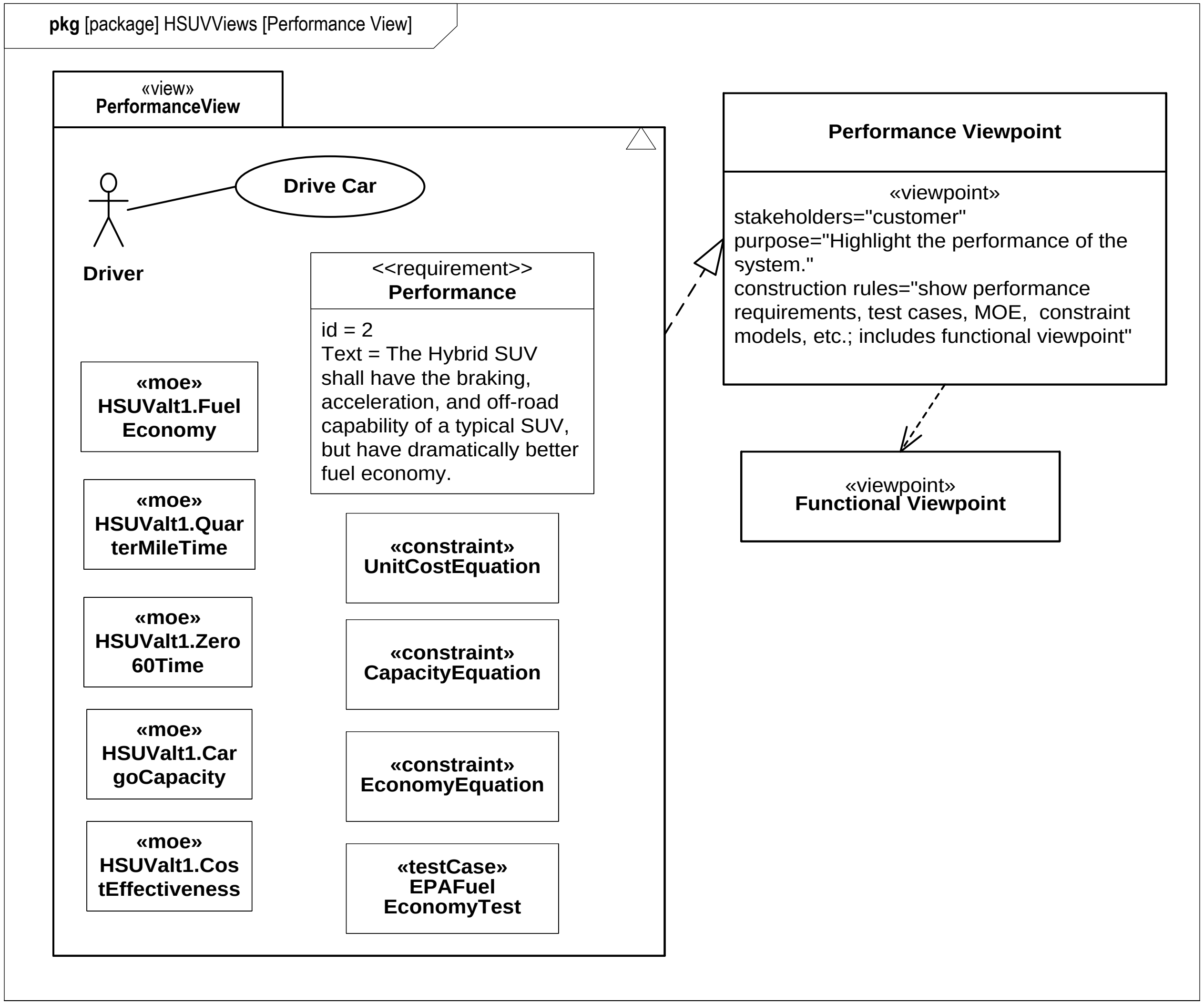
- Includes stakeholders and their concerns as fundamental element
- The environment determines the boundaries that define the scope of the system of interest relative to other systems
- Viewpoints establish the conventions by which a view is created, depicted, and analyzed
- Views conforms to a viewpoint, and addresses concern(s) of the stakeholders through a model



*text: IEEE-1471-2000 ; Maier et al. (2004); Maier (2009)*
*Graphics: IEEE-1471-2000*

# Views and Viewpoints

- **Viewpoint** represents stakeholders, their concerns, purpose, intent, and construction rules for specifying a view
- **View** is a read only mechanism that captures the model subset that addresses the stakeholder concerns
  - Realizes the viewpoint
  - Relationships between model elements established in model and not between views



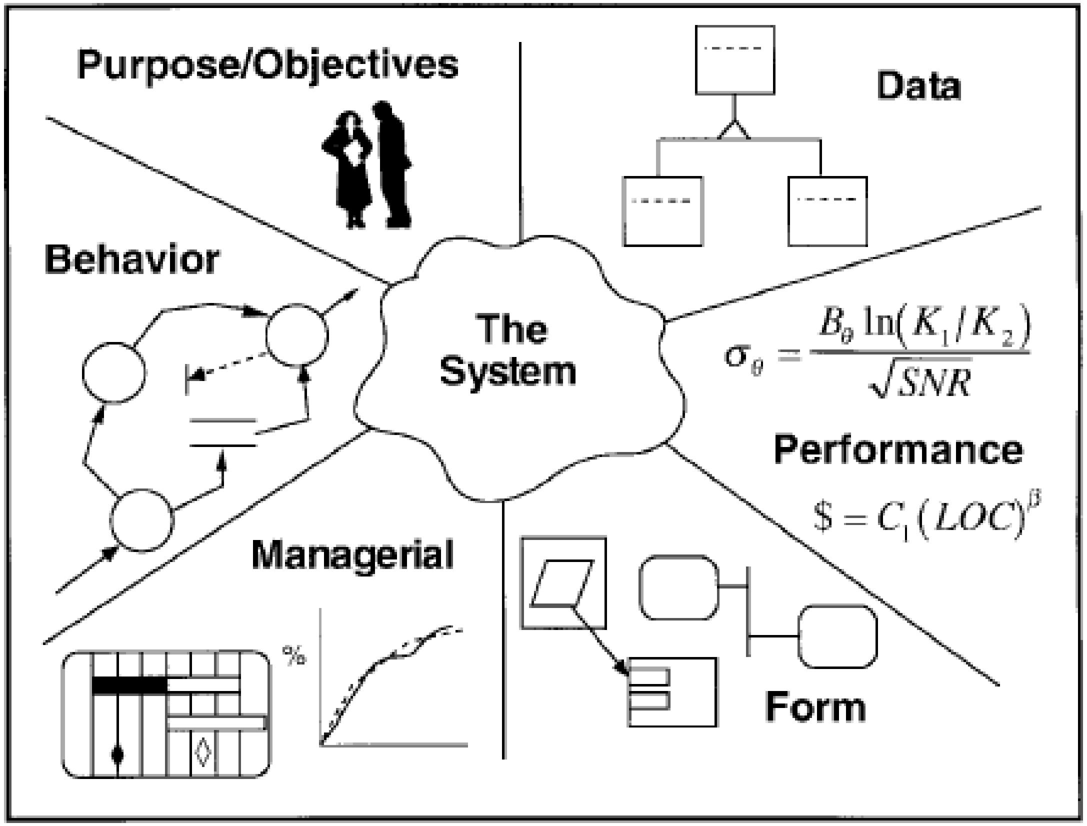
Source: “INCOSE Evaluation: Systems Modeling Language (SysML),” SysML Submission Team (SST),13, 15, 20 December 2005

# System/Architecture “Views”

**Purpose/Objective**: What the client wants

**Behavioral** (or functional): What the system does

**Managerial**: The process by which the system is constructed and managed



**Data**: The information retained in the system and its interrelationships

**Performance** (objectives or requirements): How effectively the system does it

**Form**: What the system is

- ◆ Each view represents an aspect of the actual system
- ◆ Each view may contain several <u>models</u> to capture information of the view

*Source: Maier (2009)*

# Relationship between Views

- Views chosen to be **independent**: each view should capture different piece(s) of information
- …But views are **linked**!
- Behavioral aspects **dependent** on form
  - ***System produces behavior only if form supports it!***
  - i.e. a car can't move without wheels

- Architect's role here:
  - ID views that are important, build and integrate
  - Integration across views

# Models: Objectives and Purpose

- Systems built to address what a client wants and has useful **purposes**

- Architect balances what the client wants

  (***desirability of purpose***)

  with what can be built.

  (***feasibility of system to fulfill that purpose***)

- Identify **prioritized** objectives (with the client)
  - Want measurable/quantifiable requirements
  - Must deal with "abstract" objectives a well

# Models: Objectives and Purpose

- Restate initial **unconstrained** requirements
- Want to ultimately have a “modeling language” emerge
- Identify **behavioral requirements** (what does the system need to do)
- Identify **performance requirements** as “measurable satisfaction models”
- Identify requirements that directly translate to **physical form**
- Characteristics and behaviors may **evolve**; some objectives to difficult to group as one of the above

# Models: Form

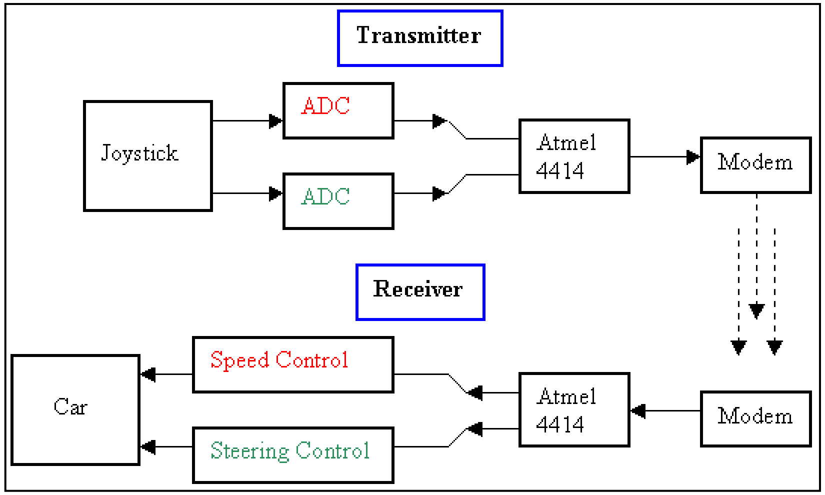
- Represents **physically identifiable** elements and interfaces of what will ultimately be built

- Includes **less tangible** issues
  - Communication protocol standards
  - Laws/regulations
  - Policies

- Degrees of **abstraction**
  - Simple exoskeleton to convey aesthetics and looks
  - Tightly coupled to performance model (i.e. model for wind tunnel test)

# Models: Form

- **Block Diagrams**
  - Must correspond to physically identifiable element of the system
    - » If not, likely more appropriate to be part of a behavioral model
  - Examples:
    - » ***System Interconnect Diagrams***: shows specific physical elements connected by physically identifiable channels; can be hierarchical



***Radio Control Car Wiring Diagram from:***
*http://www.electrokits.com/Electric-RC-Cars/RC-Car-Controller-Project*

# Models: Form

- **Block Diagrams (cont'd)**
  - **Data flow logic**: who controls the flow?
    - » Important for interfacing to disciplines
    - » System activities provide information needed to enable software architecting *(notions of software concurrency and synchronization driven by data flow discussed in later modules)*

- » ***Soft Push***: sender sends, receiver must be waiting to accept
- » ***Hard Push***: act of sending interrupts the receiver, who must accept
- » ***Blocking Pull***: receiver requests data and waits until the sender responds and sends
- » ***Nonblocking Pull***: receiver requests data and continues on without it while waiting for the sender to respond and send
- » ***Hard Pull***: receiver requests data, which interrupts the sender who must respond
- » ***Queuing Channel***: sender pushes data to a "channel" where it is stored; receiver pulls from the channel store; no one is interrupted

# Models: Behavioral/Functional

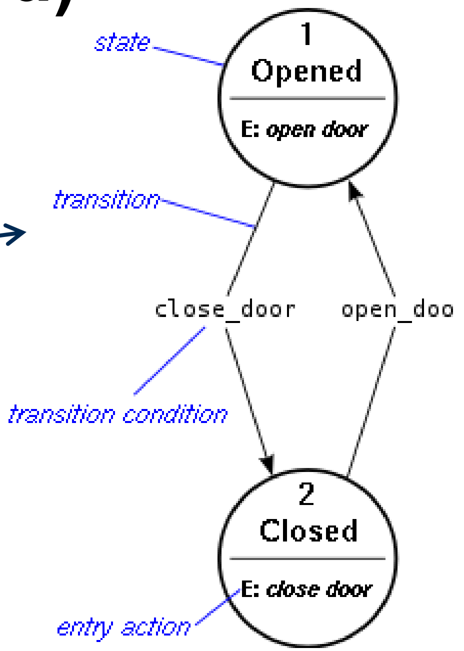
- Describes pattern of behavior
- What the system does as opposed to what the system is
  - What the system does: models of behavior
  - What the system is: models of form
- Can not always look at a scale model (of form) and infer behavior

*I can infer behavior from this form*

*I can not necessarily infer behavior from this form*

# Models: Behavioral/Functional

- **Data and Event Flow Networks (cont'd)**
  - Examples
    - Data Flow Diagram
    - Finite state machine description
    - **Functional Flow Block Diagram**



  - FFBD root principles
    - Functions decomposed hierarchically
    - Decomposition hierarchy defined graphically
    - Data elements decomposed hierarchically and defined
    - Functions are data triggered
    - Defined model structure avoids redundant definition

# Models: Performance

- Predicts how **effectively** an architecture satisfies some objectives, either functional or not
- “**Non-functional” requirements**: they do not explicitly define a functional thread of operation
- Usually **quantitative** and measurable
- Describe **system level functions**: properties possessed by the system as a whole
- Must **constrain** system behavior and form to develop a quantitative performance model

# Models: Performance

- **Analytical**

  Lower level system parameters and a mathematical rule of combination that predicts the performance parameter of interest from lower level values

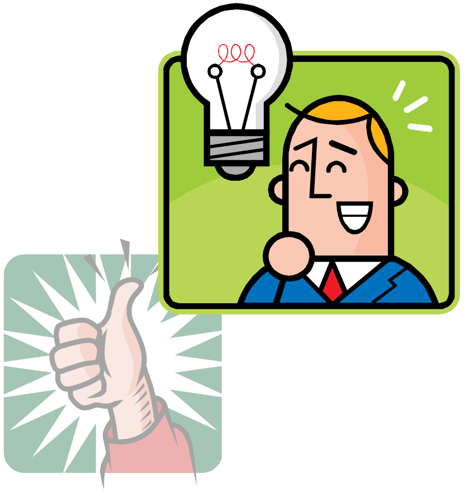
- **Simulation**

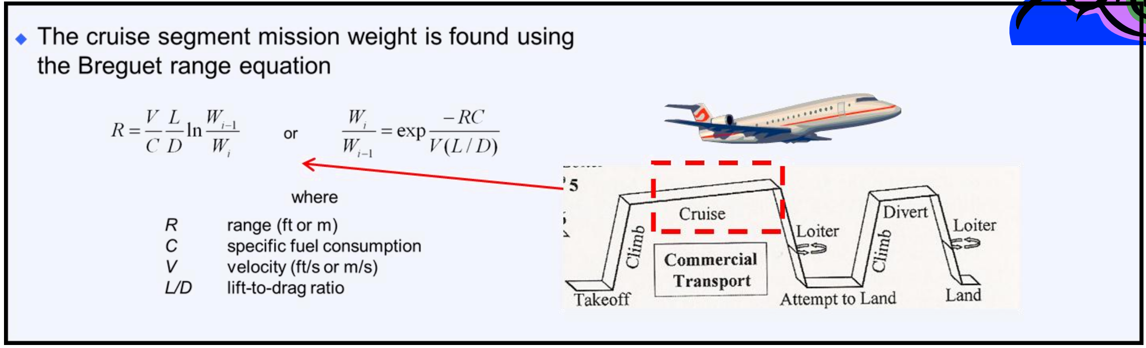
  May be used when performance may not be predicted through closed form analytical models, but more complex and difficult to explicitly identify

- **Judgment**

  Used when analytical or simulation models are inadequate or infeasible

  Human judgment captured as design heuristics





- The cruise segment mission weight is found using the Breguet range equation

$$R = \frac{V}{C}\frac{L}{D}\ln\frac{W_{i-1}}{W_i} \quad \text{or} \quad \frac{W_i}{W_{i-1}} = \exp\frac{-RC}{V(L/D)}$$

where

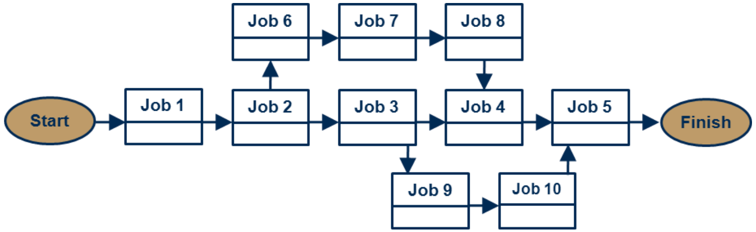
| | |
|---|---|
| *R* | range (ft or m) |
| *C* | specific fuel consumption |
| *V* | velocity (ft/s or m/s) |
| *L/D* | lift-to-drag ratio |

# Models: Data

- Data may be a part of the architecture
- Defines the data that the system itself retains, and how the relationship among the data is developed and maintained
- Data models have origins in software development and database development
- The need to find structure and relationships in large collections of data will be determinants of the system architecture

# Models: Managerial

- Milestones, budgets, and schedules may be as important to the architect as the technical effort
- Managerial view describes the **process of building** the physical system, and tracks events as they occur
- Models that comprise this view are standards in project management
  - Critical Path Methods/PERT
  - Cost and schedule metrics



- Architect will use these to monitor processes as systems is developed to **ensure integrity**

# Systems of Systems Architecting Considerations

# Systems of Systems Architecting

- **Systems of systems architectures concerned with architectures of systems created from other autonomous systems**
- **System architectures**
  - Concerned with people, activities, and technologies that make up an autonomous system within an enterprise*
  - Includes structures and behaviors
  - Autonomous systems may interact with other autonomous systems within an enterprise
  - Autonomous systems' core functionality not dependent on other autonomous systems within an enterprise

***Enterprise**: an association of interdependent organizations and people, supported by resources, which interact with each other and their environment to accomplish their own goals and objectives and those of the association

*Source: Cole, in Jamshidi (2009)*

# Systems of Systems Architecting

- **Systems of systems architectures concerned with architectures of systems created from other autonomous systems**
- **Enterprise architectures**
  - Concerned with organizational resources and activities
  - Includes people, information, capital, physical infrastructure
  - Consideration of constituent (autonomous) system characteristics within the focus of the SoS architect
  - Design of constituent (autonomous) systems not the focus of the SoS architect
  - SoS architect may consider multiple enterprises

*Source: Cole, in Jamshidi (2009)*

# Systems-of-Systems Architecture

- The management of relations between the system components is an **architectural** issue which does not belong to individual systems, but shared by all the involved components

- SoS architecture acts as a framework that directs the interaction of components with their environment, data management, communication, and resource allocation

- The system-of-systems architecture defines the interfaces and composition which guides its implementation and evolution

*Allocation of functionality to components and inter-component interaction, rather than the internal workings of individual components*

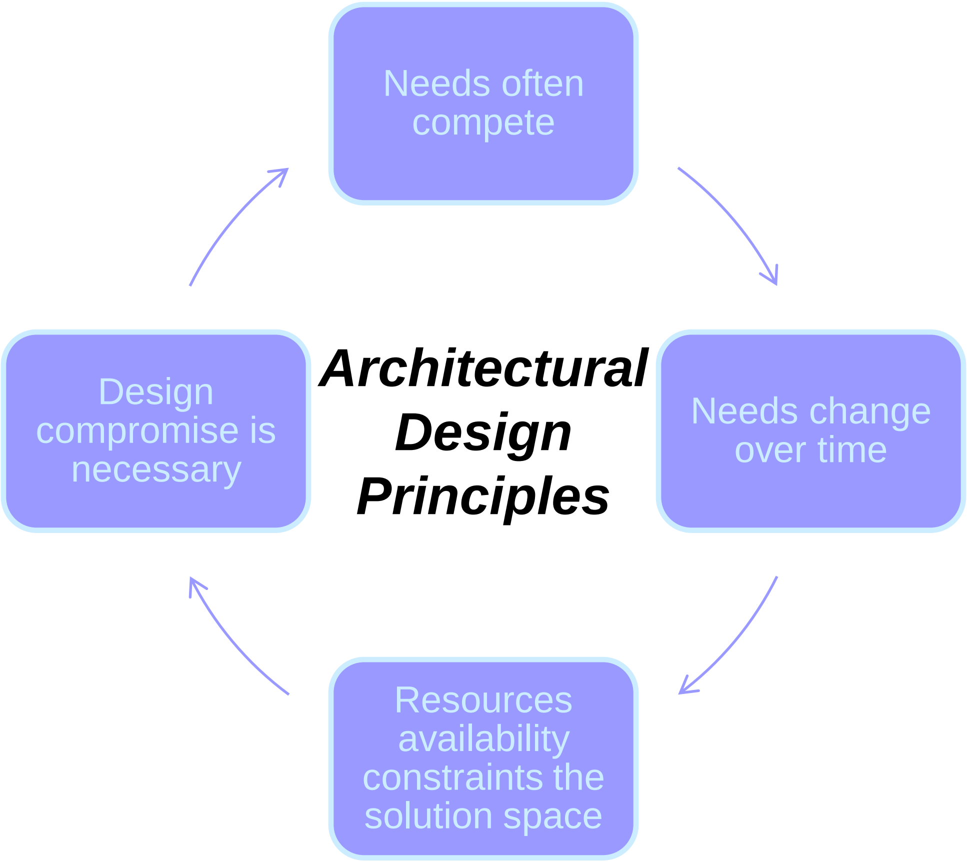
# Systems-of-Systems Architecture

*Manners by which systems and capabilities are related in a system-of-systems*

- **Structure**: Two systems are structure-related if one is a component or basis of the other.
- **Function**: Two systems are function-related if one system requires certain functions or services by another system to perform its own function.
- **Information:** Two systems are information-related if requirements or information is exchanged between the two.
- **Operation**: Two systems are operation-related if they are both used in an operation scenario to jointly fulfill a mission.
- **Generation**: Two systems are generation-related if one system will be a replacement of the other.

***Relations are determined by the interfaces between systems***

# SoS Architecture Considerations



***Architecting of a SoS warrants special considerations***

- **Autonomy**
- **Diversity**
- **Integration strategy**
- **Data architecture**
- **System protection**

*Source: Cole, in Jamshidi (2009)*

# Autonomy

- Elements of the SoS are autonomous systems
- Each has its own
  - Stakeholders
  - Mission
  - Management
  - Budget
  - etc…
- SoS integration cannot compromise the integrity of the constituent systems…**autonomy must be maintained after SoS integration**
- **If autonomy of individual systems is disrupted for the benefit of the SoS, it must be re-established**

*Source: Cole, in Jamshidi (2009)*

# Autonomy

- **Technical autonomy**
  - Integrity of external **interfaces** (of constituent systems) must be maintained
  - Integrity of **infrastructure** must be maintained
    - Unplanned infrastructure improvements on the SoS level may disrupt technical autonomy at the system level
- **Operational Autonomy**
  - Related to organizations and business processes
  - Organizations structured to operate and sustain systems using organic business processes
  - The “heart” of the operational architecture of each system, and must have autonomy

*Source: Cole, in Jamshidi (2009)*

# Complexity

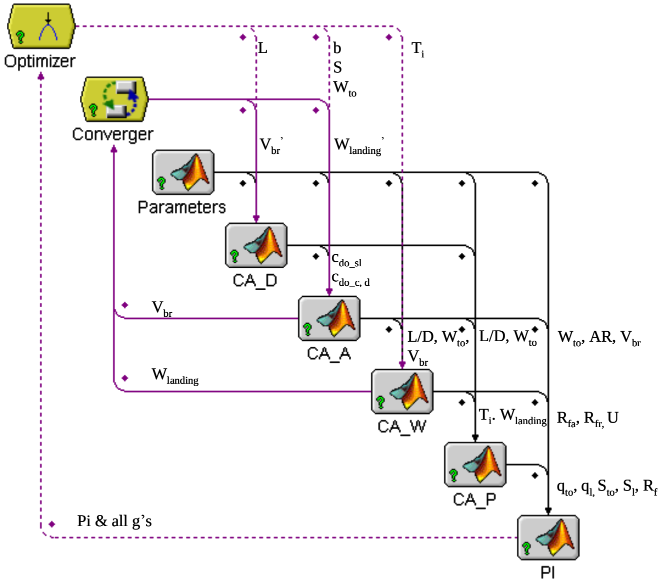
- **The existing system "tax"**
  - Complexity introduced when using existing systems to create SoS solutions
  - Using existing systems to assemble an SoS is a good starting point, but constrains the solution
  - Infrastructure used to support a system may be of little value at the SoS level (i.e. introduce complexity)
- **Natural specialization**
  - Individual systems will want to optimize to perform their primary function
  - Will likely "sub-optimize" for individual systems, which may introduce other constraints

*Source: Cole, in Jamshidi (2009)*

***SoS Architecture Considerations***

# Complexity

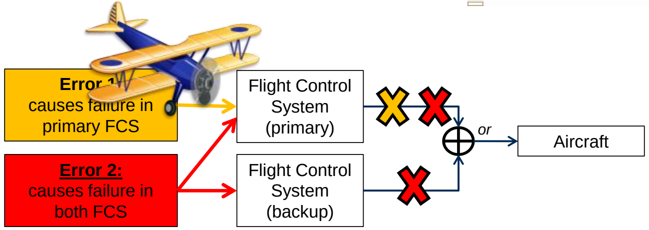
- **Natural specialization (cont’d)**
  - Must “bridge” the optimization across system, which introduces complexity



- **Fuzzy functional architecture partitions**
  - The gaps and overlaps in functional responsibilities
  - Preserving technical autonomy means multiple systems within the SoS will perform similar (or identical functions)

*Source: Cole, in Jamshidi (2009)*

# Diversity


**Error 1**
causes failure in primary FCS

**Error 2:**
causes failure in both FCS

Flight Control System (primary)

Flight Control System (backup)

*or*

Aircraft

- **Diversity reduces Common Node Failure weakness**
- Challenge: **diversity of needs**
  - – Constituent systems <u>motivated</u> by individual needs which change over time
  - – Evolving business case(s): <u>evolving</u> stakeholder needs changes each “evolutionary path”
- Challenge: **environmental diversity**
  - – Constituent systems <u>managed</u> separately
  - – Forces that shape evolution (budget, politics, leadership)

*Source: Cole, in Jamshidi (2009)*

# Integration Strategy

- At the system level…
  - Systems usually **partitioned** into elements having their own responsibilities within that system
  - Elements usually designed to be **integrated within** that system

- SoS made up of autonomous systems not originally designed as part of a component in a larger system (or that SoS)

*Source: Cole, in Jamshidi (2009)*

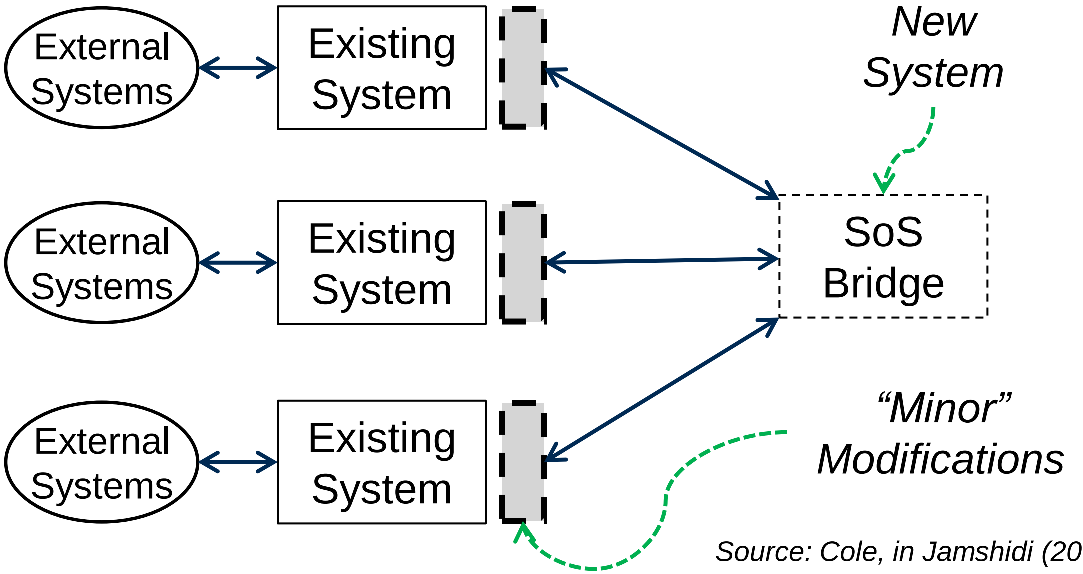
# Integration Strategy

- Integration issues
  - **Physical integration:** do all the systems use compatible interface protocols?
  - **Functional integration**: are the various functions performed by each system de-conflicted?
    - » **Isolation**: isolating the functions performed by one system within the SoS from those performed by other systems
    - » **Damping**: muting certain functions to allow systems to work together
  - **Semantic integration**: are data and signals commonly interpreted by the different systems?

*Source: Cole, in Jamshidi (2009)*

# Integration Strategy

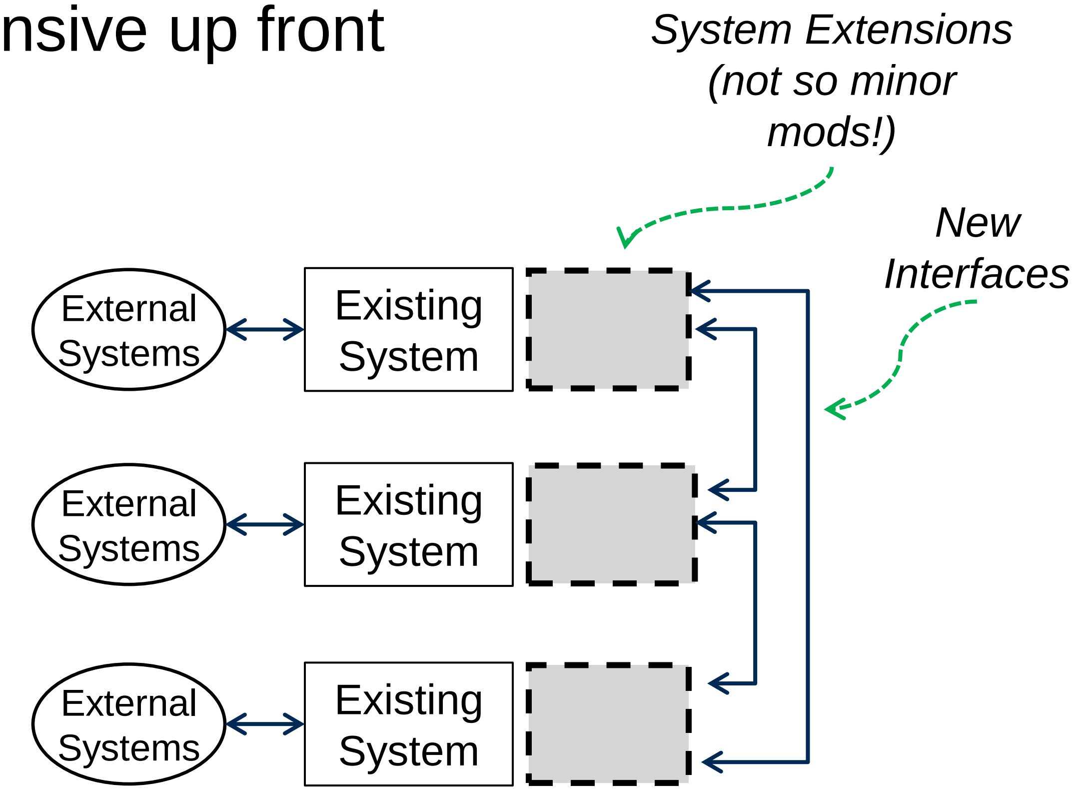
- **Solution: SoS Bridging**
  - Introducing a new system that has the responsibility of dealing with physical, functional, and semantic integration…acts as a “bridge”
  - Minimizes modification to existing systems
  - Less expensive up front
  - Burdensome to operations and adds complexity
  - Most common



Source: Cole, in Jamshidi (2009)

# Integration Strategy

- **Solution: SoS Refactoring**
  - Easier to operate and less complex that bridging
  - More disruptive to individual systems
  - More expensive up front



*System Extensions (not so minor mods!)*

*New Interfaces*

*Source: Cole, in Jamshidi (2009)*

# Data architecture

- SoS needs regarding data architecture
  - Data consistency and semantics
  - Persistent storage of shared data
    - ***Data may be owned by one system, but needed across the SoS***

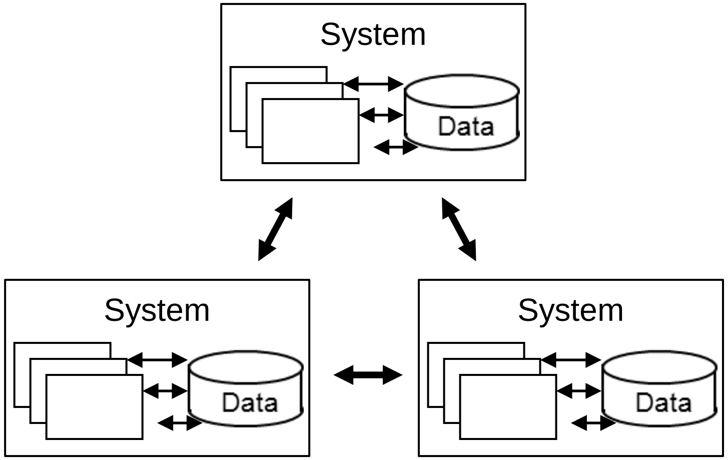
- Single data store as an option
  - Low complexity
    - Low risk in terms of data integrity
    - Low expense to create and manage
  - Limit practicality
    - Does not preserve autonomy of existing systems
    - Difficult to meet required performance and availability

*Source: Cole, in Jamshidi (2009)*

# Data architecture

## ◆ Uncoordinated Data Model

- Simple and economical strategy
  - » Requires shared data be **exchanged via traditional interfaces** between systems
  - » Requires each system **independently deal with data structure** and semantic problems
- Problems with data structure and semantics introduce risks
- Potential for high volume of duplicate data
- Good if SoS exchanges low volume of data

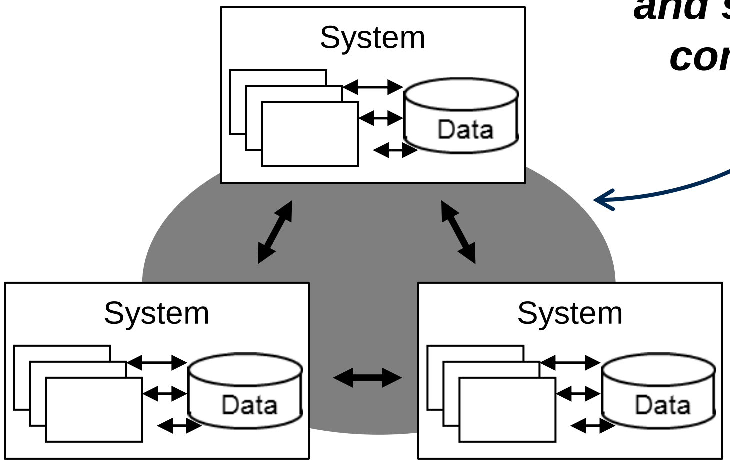


*Source: Cole, in Jamshidi (2009)*

# Data architecture

## ◆ Coordinated Data Model

- Mitigates the semantic problem found in the uncoordinated data model
- Agreement between the system coordinates data format and semantics
- Maintains simplicity of the uncoordinated model



***Agreement to coordinate naming, structure, and semantics of common data***

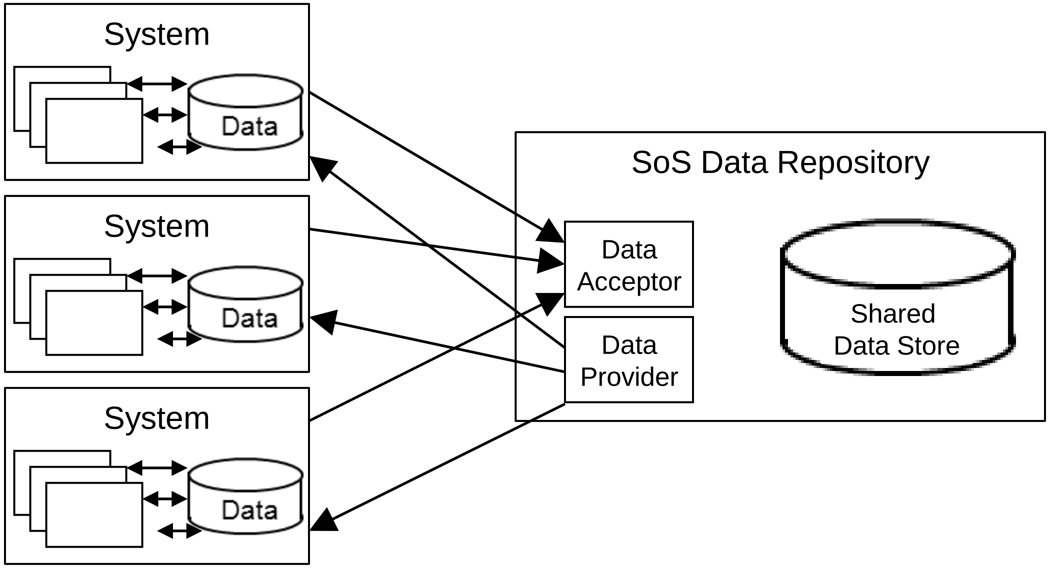
*Source: Cole, in Jamshidi (2009)*

***SoS Architecture Considerations***

# Data architecture

## ◆ Federated Data Model

- Most sophisticated approach
- Best applied when there is a large amount of data shared
- Only approach that has a **separate SoS data store** outside of the existing systems
- **Repository** contains the shared data
- **Data owned by a system, posted to repository into an agreed to format**



*Source: Cole, in Jamshidi (2009)*

*SoS Architecture Considerations*

# System Protection

- **Security** involves allowing systems to interact while preventing unauthorized access to system data and resources

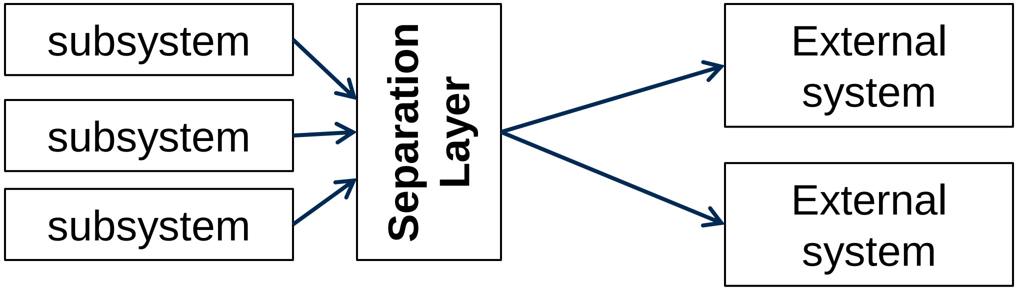
- Key objectives (and terminology) of **security**
  - **Confidentiality**: prevent unauthorized access
  - **Authentication**: provide a means or identifying authorized users
  - **Integrity**: restrict unauthorized modifications to resources
  - **Nonrepudiation**: guarantee identities of resource consumers and providers

*Source: Cole, in Jamshidi (2009)*

## *SoS Architecture Considerations*

# System Protection

- **Unintentional disruption** by other systems within the SoS is the other side of protection
  - Other systems may overload a system that provides a critical function
  - Fault in one system may ripple throughout the SoS
  - System isolation employed for protection against such disruptions
    - » ***Introduces a separation layer between internal subsystems of a system and external systems***



*Source: Cole, in Jamshidi (2009)*

# Success Factors

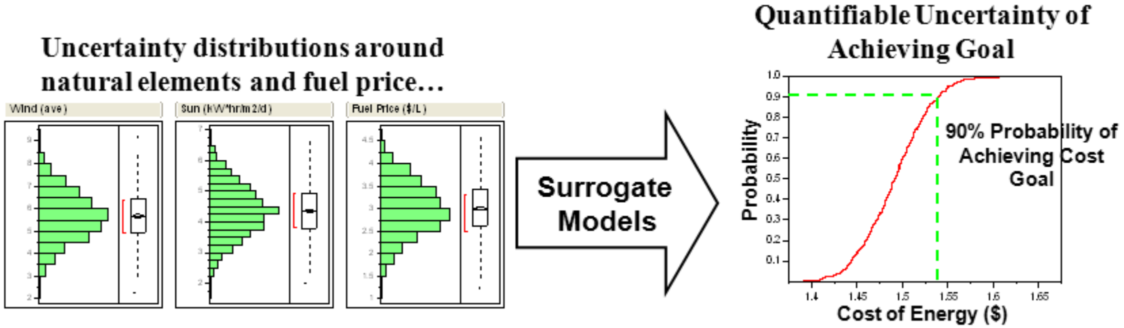
- Recommended architecture related factors contributing to the success of the SoS
- Concepts apply to single systems
- Especially important to SoS!

  - **Robust design**
  - **Architecture alignment**
  - **Architecture governance**
  - **Architecture description**

*Source: Cole, in Jamshidi (2009)*

# Robust Design

- Robust designs are those that meet requirements **consistently** and are insensitive to small changes in uncontrollable variables
- Serve their intended purpose under **full range** of environmental conditions
- Wide single system robust design body of knowledge
- Unique aspects to **SoS architecture robustness** given that the constituent systems are diverse and need to maintain autonomy



*Source: Cole, in Jamshidi (2009); and Ender et al (2010)*

**_SoS Architecture Success Factors_**

# Robust Design

## Business Case Robustness

- Needs change over time, which changes constituent systems' roles in the SoS
- SoS functions should be insensitive to changes in business case for each system in the SoS

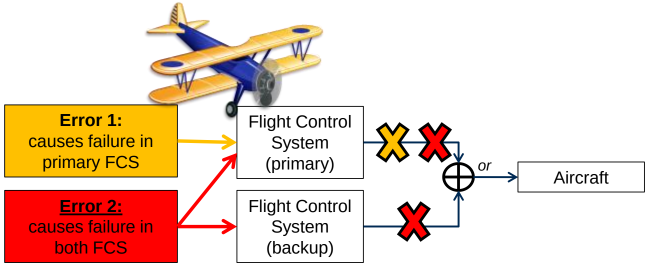
## Technological Robustness

- Related to the technological environment
- Desire insensitivity to changes in the technologies themselves within the SoS

*Source: Cole, in Jamshidi (2009)*

# Robust Design

- **Schedule Robustness**
  - Ability of a system to provide necessary capability to an SoS **on time**
  - System improvements may be delayed for technical or financial reasons
    - If that system provides the sole source of a critical capability, system is not schedule robust
    - If there is a **contingency approach** to meeting that critical capability, system is schedule robust (Redundancy? Diversity?)



**Error 1:** causes failure in primary FCS

**Error 2:** causes failure in both FCS

Flight Control System (primary)

Flight Control System (backup)

*or*

Aircraft

*Source: Cole, in Jamshidi (2009)*

***SoS Architecture Success Factors***

# Architecture Alignment

- Very probable that creating, improving, or otherwise manipulating an SoS will introduce **disruption to autonomy** of constituent systems

- Must expect disruption in this case and plan to realign and reestablish constituent systems
  - – Realign **organizations** to function within the updated SoS context
  - – Update **business processes** and procedures to function within the updated SoS context
  - – Realign **technological** aspects
    - » Easier said then done!

*Source: Cole, in Jamshidi (2009)*

*SoS Architecture Success Factors*

# Architecture Governance

- Changes among autonomous systems should be coordinated within the SoS
- Constituent systems must honor a common set of rules for functions across systems (within the SoS) which form the basis for architecture governance

*Source: Cole, in Jamshidi (2009)*

# Architecture Governance

- **Governance roles and responsibilities**
  - Deals with “fuzzy partition” of a system’s role in the SoS as its needs change over time
  - Coordinated changes occur within the context of managing roles and responsibilities

- **Interface governance**
  - Deals with interfaces between systems (also “fuzzy”)
  - Systems that share data must coordinate changes to the data structure itself

*Source: Cole, in Jamshidi (2009)*

***SoS Architecture Success Factors***

# Architecture Description

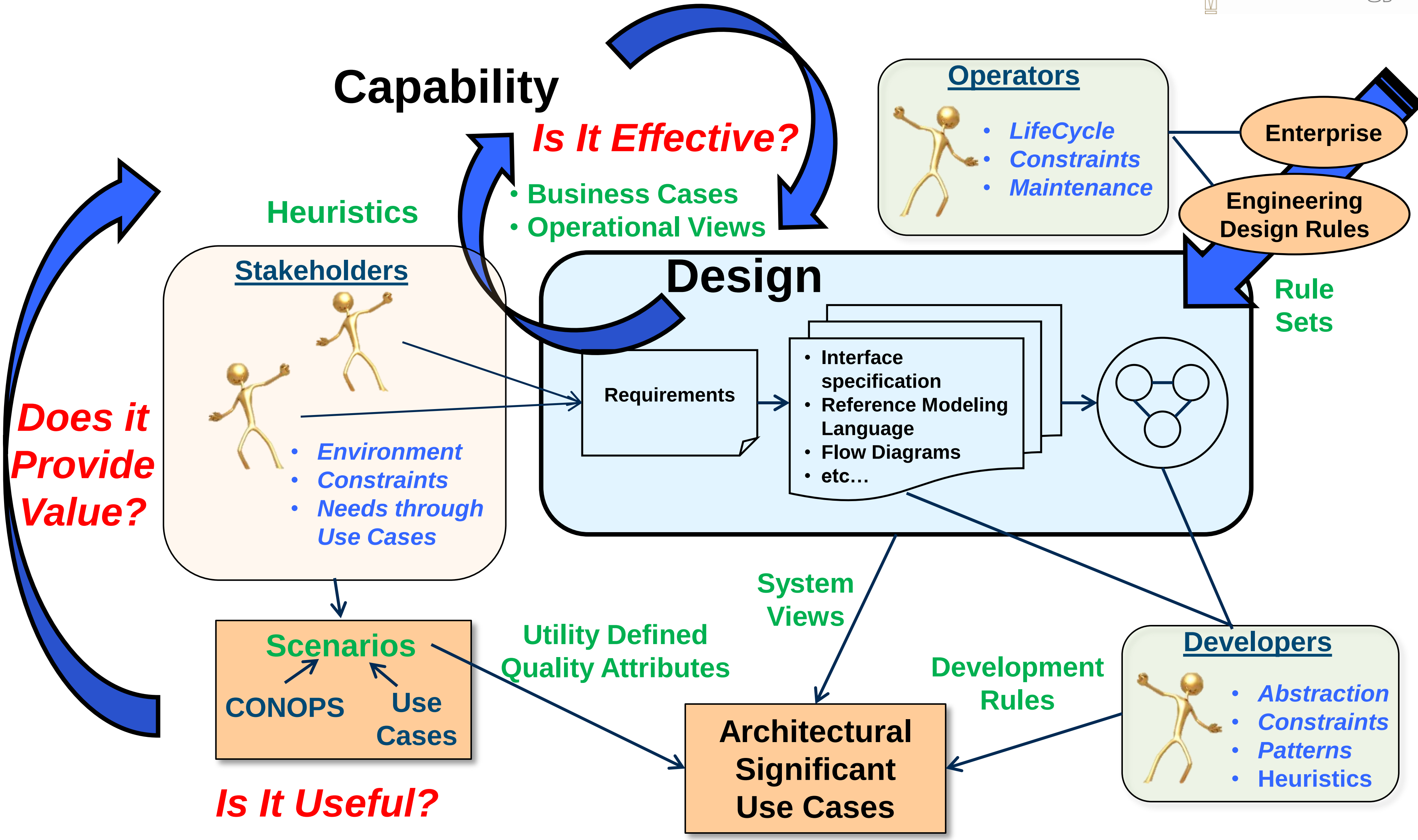
- Becomes important to represent the architecture of increasingly complex systems using a well defined model
- **Architecture model** provides means for
  - performing analysis of system structure and behavior
  - describing an implementation plan
  - describing the architecture as roles are spread across many engineers/stakeholders
- **Architecture descriptions** assembled through multiple viewpoints
- **Architecture frameworks** provide that roadmap for describing the system architecture

*Source: Cole, in Jamshidi (2009)*

# Leadership and Management:

# The Role of the Systems Architect

# Perspective of the Systems Architect



**Capability**

***Is It Effective?***

- Business Cases
- Operational Views

Heuristics

**Stakeholders**

- *Environment*
- *Constraints*
- *Needs through Use Cases*

**Operators**

- *LifeCycle*
- *Constraints*
- *Maintenance*

Enterprise

Engineering Design Rules

Rule Sets

**Design**

**Requirements**

- Interface specification
- Reference Modeling Language
- Flow Diagrams
- etc…

***Does it Provide Value?***

Scenarios

CONOPS

Use Cases

***Is It Useful?***

Utility Defined Quality Attributes

System Views

**Architectural Significant Use Cases**

Development Rules

**Developers**

- *Abstraction*
- *Constraints*
- *Patterns*
- Heuristics

# Phases of Architecting

## Changes as project moves from phase to phase

| **Early** | **Mid** | **Completion** |
|---|---|---|
| Structuring of the unstructured (need, solutions, technical possibilities) | Integration of competing (sub)systems and interests | Certification that systems is suitable for use |
| *Art* | *Rational and Normative* | *Art and Science* |
| Narrative Form | Specific Form | Narrative and Measured Forms |

# Language of the Architect

## Changes as project moves from phase to phase

**Early**

- Heuristics
- Stories
- Con-ops
- Scenarios

*Narrative, Visual*

Narrative Form

**Mid**

- Requirements
- Behavior
- Structure
- Function
- Rules

*Visual, Functional*

Specific Form

**Completion**

- Performance
- Analysis
- Evaluation
- Utility

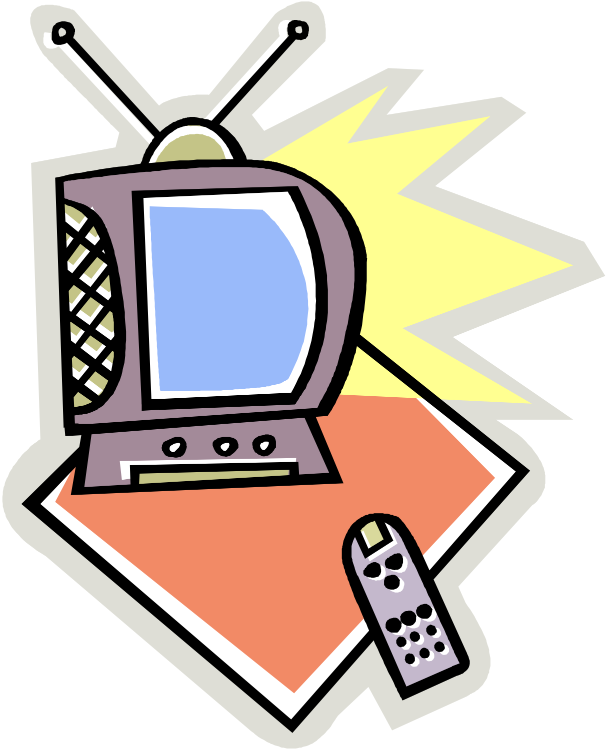
*Participative*

Narrative and Measured Forms

# The Narrative Form

# Need vs. Requirement vs. Utility

- Need:
  - Something that solves a perceived problem or desire; or perceived market
  - Responds to an opportunity
- Requirement
  - Need expressed in engineering terms
  - Analysis conducted to validate need versus system capabilities
  - Is testable
- Utility
  - Evaluation of product vs. need
  - Is testable
  - May not reflect requirement set

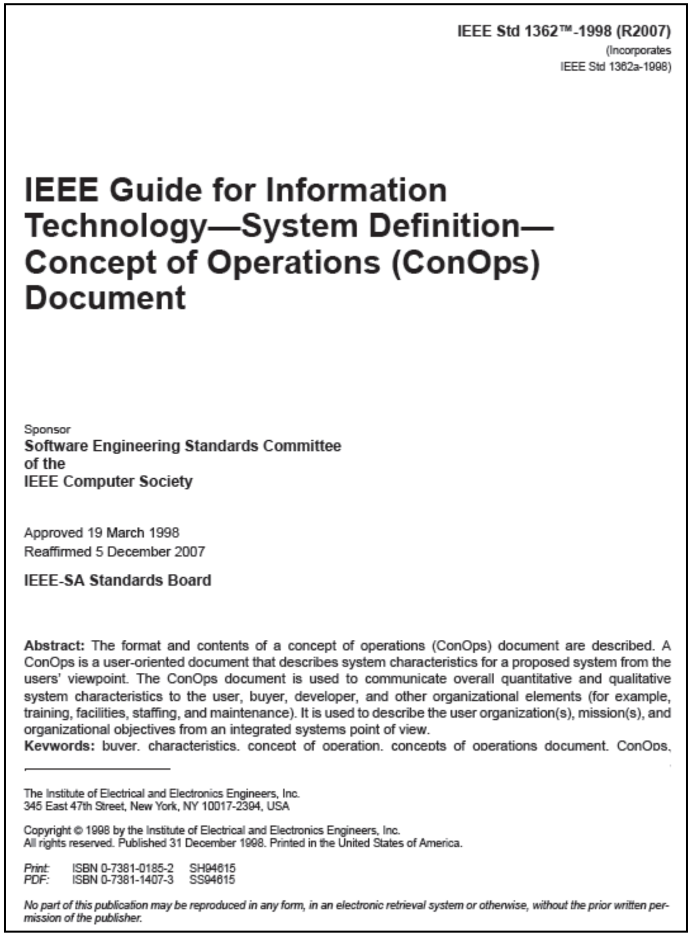


# Concept of Operations

- Create, visualize and discuss use scenarios in complex environments; Used as a strategic planning tool to reduce chance of overlooking important factors; provides balanced perspective
- Explore scenarios for clear understanding of operational needs and performance requirement rationale

# Concept of Operations (CONOPS)

- A user oriented document that describes system **characteristics of the to-be-delivered system from the user's viewpoint**
- Used to communicate **overall quantitative and qualitative system characteristics** to the user, buyer, developer, and other organizational elements (e.g., training, facilities, staffing, and maintenance)
- Describes the **user organization(s), mission(s), and organizational objectives** from an integrated systems point of view



IEEE Std 1362™-1998 (R2007)
(Incorporates
IEEE Std 1362a-1998)

**IEEE Guide for Information Technology—System Definition—Concept of Operations (ConOps) Document**

Sponsor
**Software Engineering Standards Committee
of the
IEEE Computer Society**

Approved 19 March 1998
Reaffirmed 5 December 2007

**IEEE-SA Standards Board**

**Abstract:** The format and contents of a concept of operations (ConOps) document are described. A ConOps is a user-oriented document that describes system characteristics for a proposed system from the users' viewpoint. The ConOps document is used to communicate overall quantitative and qualitative system characteristics to the user, buyer, developer, and other organizational elements (for example, training, facilities, staffing, and maintenance). It is used to describe the user organization(s), mission(s), and organizational objectives from an integrated systems point of view.
**Keywords:** buyer, characteristics, concept of operation, concepts of operations document, ConOps,

---

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1998 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 31 December 1998. Printed in the United States of America.

*Print:* ISBN 0-7381-0185-2 SH94615
*PDF:* ISBN 0-7381-1407-3 SS94615

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

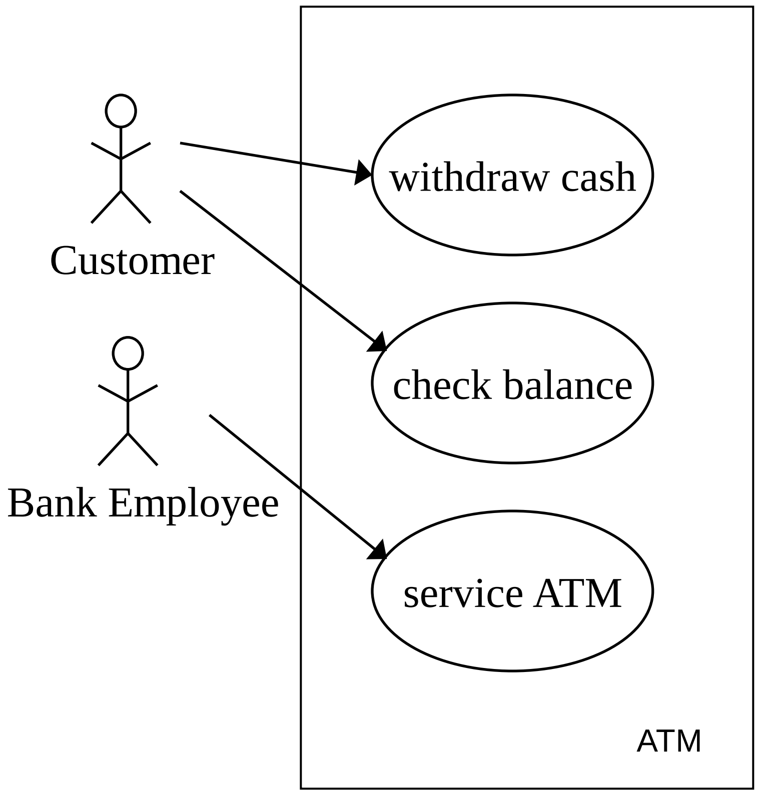
Source: IEEE Std 1362-1998

# What is a Use Case?

- Describes the desired behavior of a system and its users
  - at a superficial level of detail
  - with “sunny-day” and “rainy-day” scenarios
  - with some generalization of the roles and activities
  - a set of activities within a system
- A Use Case is the set of scenarios that provides positive value to one or more external actors
  - actors are the people and/or computer systems that are outside the system under development
  - scenarios are dialogs between actors and the system
  - no information about the internal design

# The UML Use Case Diagram

- In UML (Unified Modeling Language), it is possible to show a picture of the system as a group of use cases:
  - each stick figure is an actor
  - each ellipse represents a use case
- The diagram is deceptively simple
  - behind each ellipse, there might be a whole bunch of scenarios – sunny-day, alternatives, failures
  - the diagram is only a "summary"



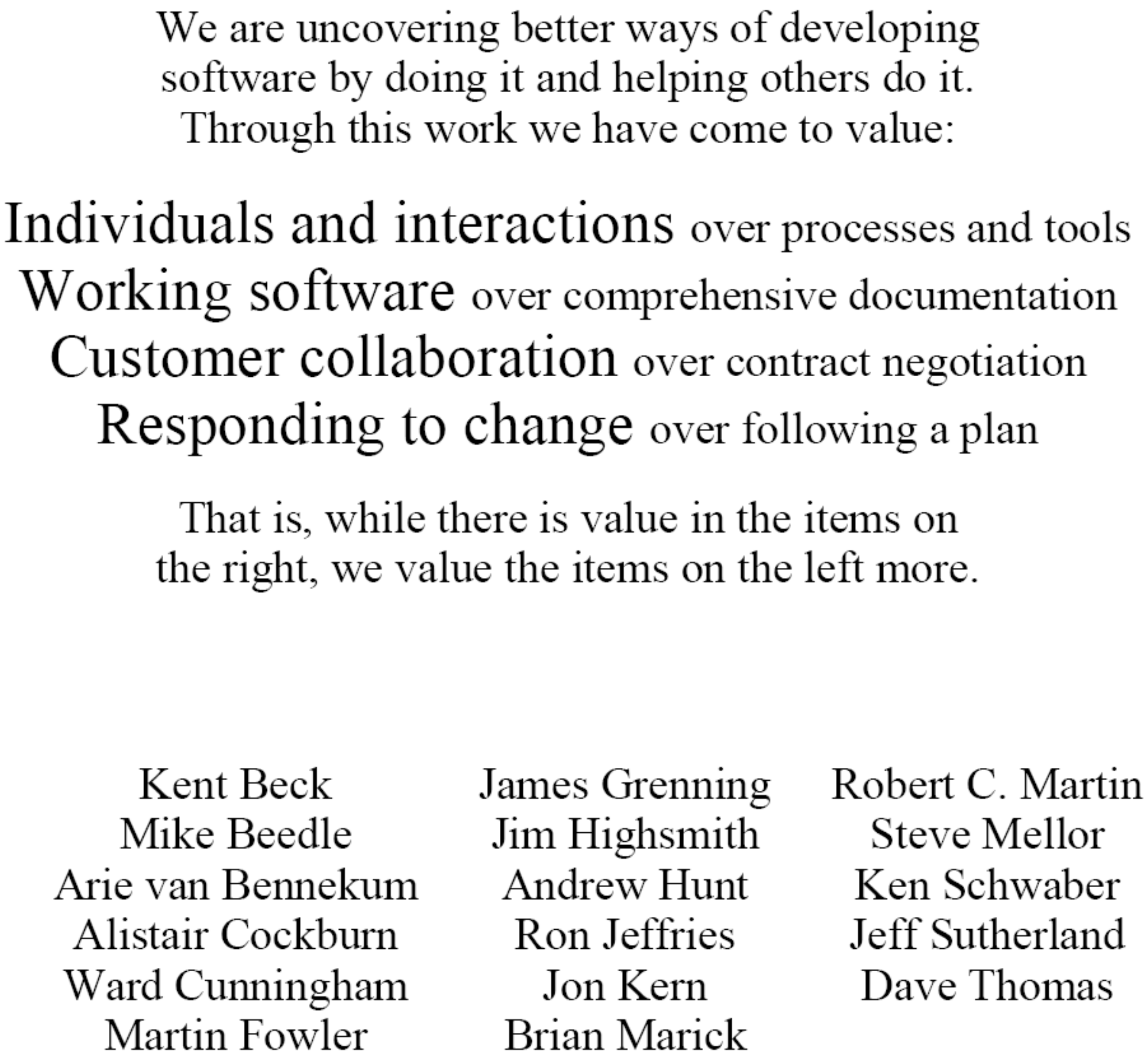
# Stories

- A story is a high-level definition of a requirement
  - Enough information so the developer can produce a reasonable estimate of the effort to implement it
  - Not so much that it requires a lengthy effort to agree on the specification of it

# What Does “AGILE” Imply?

- Agile:
  - quick and well-coordinated in movement; lithe
  - marked by an ability to think quickly; mentally acute or aware
  - characterized by quickness, lightness, and ease of movement; nimble
- Agile Software Development:
  - a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between customer and self-organizing, cross-functional teams

# The Agile Manifesto



We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

**Individuals and interactions** over processes and tools
**Working software** over comprehensive documentation
**Customer collaboration** over contract negotiation
**Responding to change** over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

# Agile Applied to Systems Engineering

- Agile development methods require a different paradigm for project management, focused on small, frequent incremental releases
- It is not clear that Agile Development methods, as developed for software programming, apply well to systems engineering
  - Agile software development assumes a mature and tested hardware baseline is available
  - Most experience is limited to IT-based systems
  - For larger complex hardware/software systems it is difficult to divide the work breakdown into 30 day incremental tasks
  - It is difficult for organizations to manage simultaneously the planning cultures of traditional development and agile development
- How do we apply agile techniques to SE?

# Scaling Agile Approaches

- Separate type of outcome
  - Tangible outcomes: physical artifacts
  - Intangible outcomes: information, including SW (not manufactured)
- Evaluate type of work
  - Inventive: result of creative input, exploratory in nature
  - Engineering: science & engineering to produce outcomes
  - Craft: repetitive tasks around work that has been done before
- These drive how you define your scheduling model and approach

Aaron J. Shenhar and Dov Dvir ,
Reinventing Project Management:
The Diamond Approach to
Successful Growth and Innovation

# N2 on Managing versus Type

| | Tangible | Intangible | Inventive | Engineering | Craft |
|---|---|---|---|---|---|
| **Tangible** | | Risk of forcing all development down same path | High risk of customer dissatisfaction | High risk of technology maturity issues | Risk of being late to market |
| **Intangible** | Use multiple development models | | High risk of customer dissatisfaction | High risk of utility or use case issues | Generally low risk unless innovation is a premium |
| **Inventive** | Build several prototypes and test with customers | Case for incremental development with frequent customer interaction | | Risk of immature requirements leading to poor use case design | Risk of disruptive design or process issues |
| **Engineering** | Evolutionary development approach with several fielded increments | Early increments focus on system use cases and utility | Use M&S to focus customer on use cases and utility | | Risk of cost or quality issues |
| **Craft** | Waterfall approach or evolutions focused on improved cost & quality | Accelerate fielded systems to evaluate utility and maturity | Early prototypes to mature processes | Early prototypes to prove technology | |

# Keys to Agile SE

- The architectural framework is at the center, and key to all other success
- Rapid development of architectural rules
- Rapid evolution of architectural quality attributes
- A model based environment for developing the architecture and evaluating applications
- Close connection between the developer and stakeholders, direct interaction in the process

# The Agile Architect

1. Deliver working solutions
2. Maximize stakeholder value
3. Find solutions which meet the goals of all stakeholders
4. Enable the next effort
5. Manage change and complexity

The Architect's primary objective is a working *solution*

*The best solution make not need significant development*

http://www.agilearchitect.org/agile/principles.htm

# The Architect’s Decisions

- Determine the Application Type
  - Services, clients, data, scientific, control, etc.
- Determine the Deployment Strategy
  - Embedded, General Purpose, Client-server, Cloud, etc.
- Determine the Appropriate Technologies
  - Execution, development, infrastructure, skills
- Determine the Quality Attributes
  - Performance, ilities, development
- Determine the Crosscutting Concerns
  - Resource management: Communication, memory, etc.
  - Exception management: safety, reliability, error capture
  - Instrumentation/data visibility

Microsoft Application Architecture Guide, 2nd Edition (Chapters 1-4)

# Architecture Concerns

Beyond the requirements document:

- How will the user experience be managed?
- How will the development be managed?
- How will the software be deployed and managed?
- How will the application support update and modification over time?
- What similar architectural trends or patterns exist that might influence development or deployment?
- What are other key quality attributes, such as security, performance, modifiability, portability, etc.?

# Key Agile Architecture Tenets Today

- Build to change instead of building to last
  - Design in flexibility for growth
- Model to analyze and reduce risk
  - Views, visualizations, modeling languages, design tools
- Use models and visualizations as a communication and collaboration tool
  - Views and visualizations for user buy-in
- Identify key engineering decisions
  - Views, design patterns, model architectures
- Use an incremental and iterative approach to refine your architecture

Microsoft Application Architecture Guide, 2nd Edition (Chapters 1-4)

# Know the Architecture Landscape

- Create User empowerment
  - Focus on the user experience
  - Allow the user to define how they interact
  - Use scenarios to design simple user interactions
- Follow market maturity
  - Take advantage of existing platform and technology options
  - Focus design on what is uniquely valuable in your application, reuse elsewhere
  - Use patterns that provide proven solutions for common problems
- Develop flexible designs
  - Loose coupling to allow reuse and to improve maintainability
  - Pluggable or service oriented designs to provide future extensibility
- Stay abreast of future technology trends
  - Information services, media convergence, device convergence, computing/networks, clouds, etc.

Microsoft Application Architecture Guide, 2nd Edition (Chapters 1-4)

# Four Architecture Principles

1. **Separation of Concerns**
   - Separate aspects of a problem
   - Minimize interaction points between modules
2. **Abstraction**
   - Build hierarchical layers of abstraction
   - Do not duplicate functions
3. **Simplicity**
   - Make it easy to understand, check, and modify
   - One function or feature (or at least a cohesive set) per module
   - Only design what is necessary
4. **Restriction of information**
   - Localization of information
   - One modules internal details hidden from other modules
   - Basic principle of object oriented design

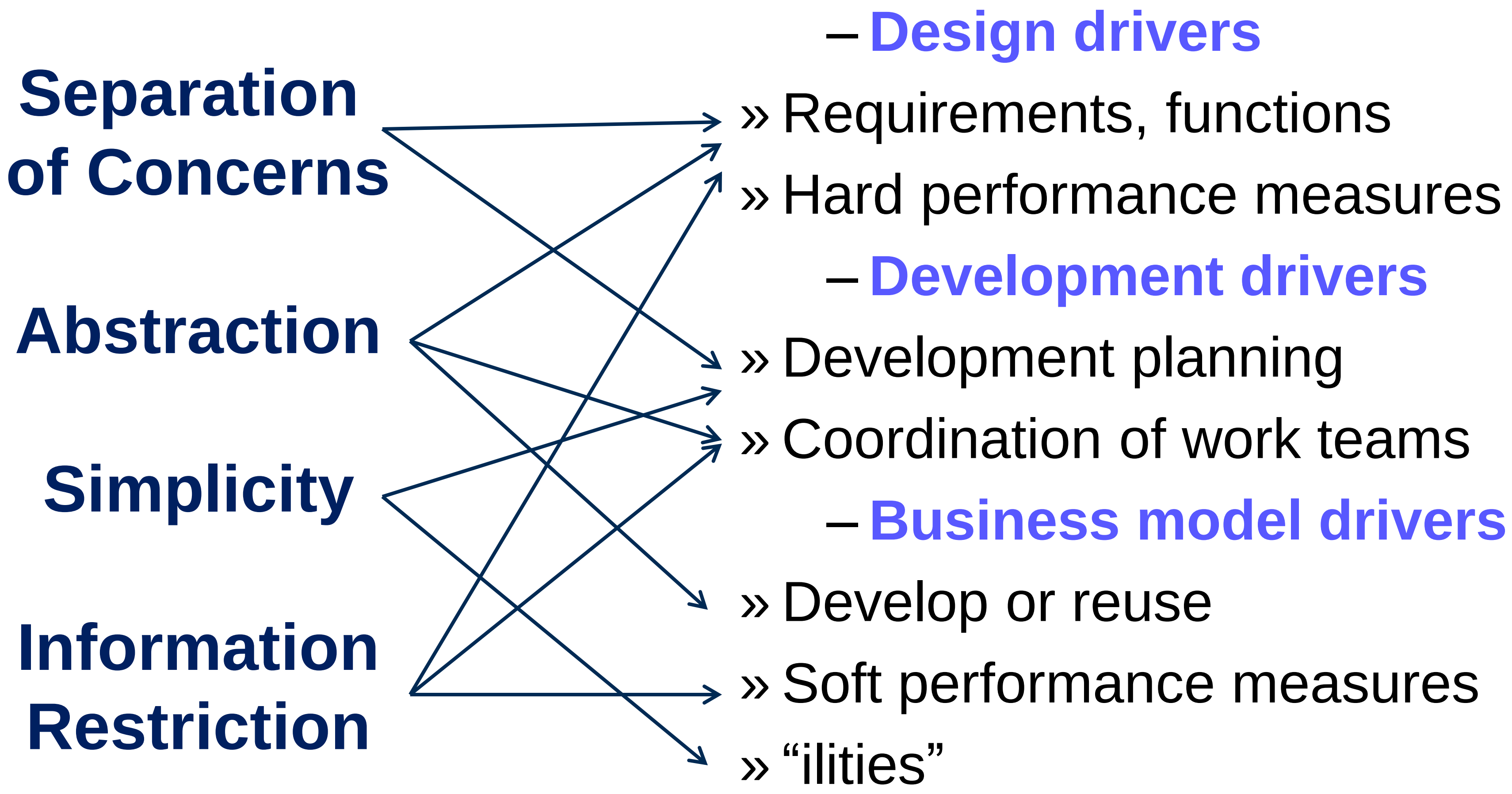
**These Scale to Anything!**

# Architectural Quality Attributes

- *How do I evaluate the quality of the architecture?*
  - **Design drivers**
    - Requirements, functions
    - Hard performance measures
  - **Development drivers**
    - Development planning
    - Coordination of work teams
  - **Business model drivers**
    - Develop or reuse
    - Soft performance measures
    - “ilities”

# Architectural Quality Attributes

- *How do I evaluate the quality of the architecture?*



Separation of Concerns

Abstraction

Simplicity

Information Restriction

- **Design drivers**
  - » Requirements, functions
  - » Hard performance measures
- **Development drivers**
  - » Development planning
  - » Coordination of work teams
- **Business model drivers**
  - » Develop or reuse
  - » Soft performance measures
  - » “ilities”

# Example Quality Factors and Architectural Methods

- Safety
- Security
- Robustness
- Resiliency
- Availability
- Portability
- Reuse
- Openness
- Modifiability
- Testability
- Maintainability

- Separation, simplicity
- Abstraction, restriction
- Distribution
- Redundancy
- Health monitoring
- Virtualization
- Encapsulation
- Standardization
- Design rules, patterns
- Partitioning
- documentation

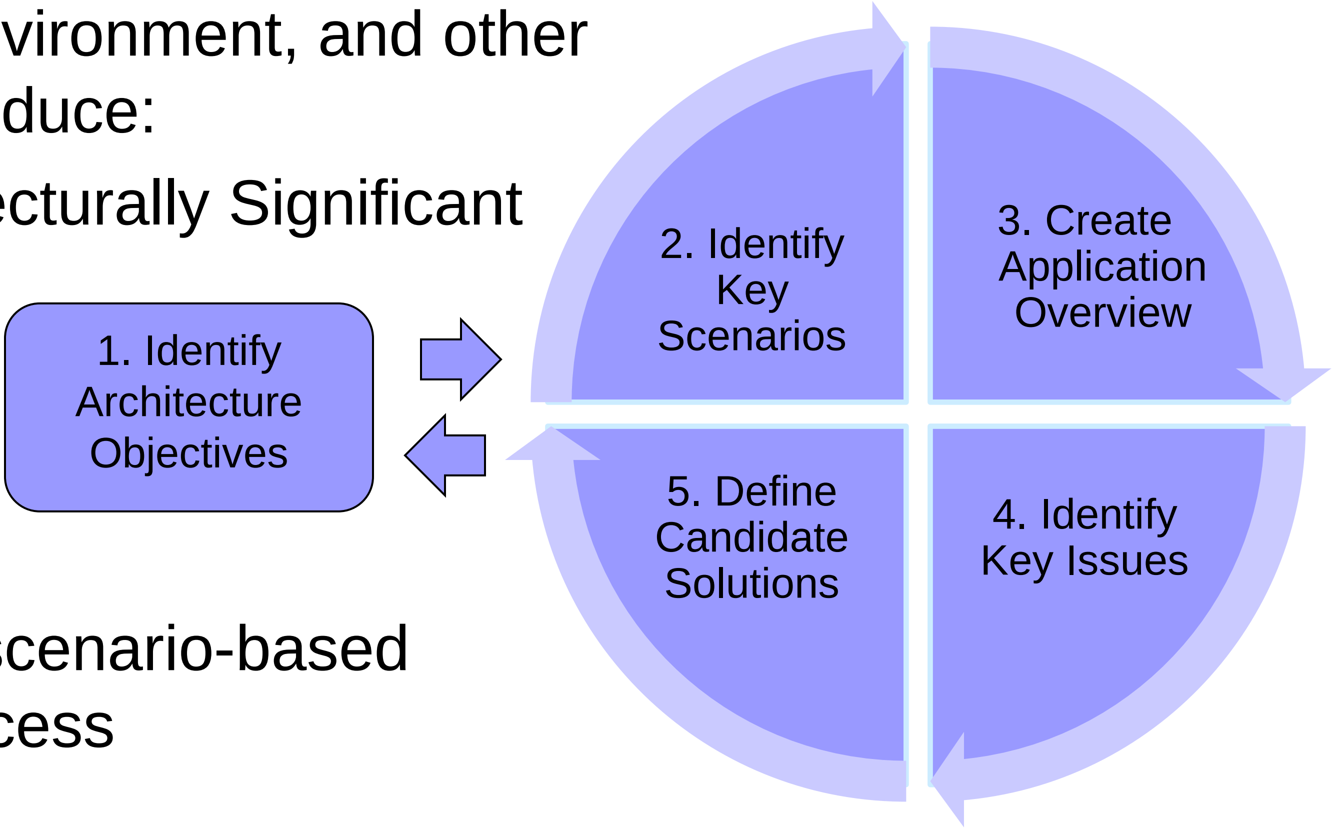
# Quality Attribute Characterization

**Each quality attribute characterization is divided into three categories: *external stimuli, architectural decisions,* and *responses.***

- ***External stimuli*** (or just stimuli for short) are the events that cause the architecture to respond or change.
- To analyze an architecture for adherence to quality requirements, those requirements need to be expressed in terms that are concrete and measurable or observable. These measurable/observable quantities are described in the ***responses*** section of the attribute characterization.
- ***Architectural decisions*** are those aspects of an architecture - components, connectors, and their properties - that have a direct impact on achieving attribute responses.

# Techniques for Architecture Evaluation

- Use cases and usage scenarios, functional requirements, non-functional requirements, technological requirements, the target deployment environment, and other constraints produce:
- A list of Architecturally Significant Use Cases
- These feed a scenario-based evaluation process



1. Identify Architecture Objectives

2. Identify Key Scenarios

3. Create Application Overview

4. Identify Key Issues

5. Define Candidate Solutions

Source: Microsoft Application Architecture Guide, 2nd Edition (Chapters 1-4)

# Techniques for Architecture and Design

1. Identify Architecture Objectives.
   - User, business, development
2. Identify Key Scenarios.
   - Use-case scenarios focus your design and allow architecture evaluation
3. Create Application Overview.
   - Identify application type, deployment architecture, architecture styles, and technologies
4. Identify Key Issues.
   - based on quality attributes and crosscutting concerns
5. Define Candidate Solutions.
   - Create an architecture prototype

Source: Microsoft Application Architecture Guide, 2nd Edition (Chapters 1-4)

# Scenario-Based Evaluation Methods

- **Software Architecture Analysis Method (SAAM)**
  - SAAM was originally designed for assessing modifiability, but later was extended for reviewing architecture with respect to quality attributes such as modifiability, portability, extensibility, integratability, and functional coverage.
- **Architecture Tradeoff Analysis Method (ATAM)**
  - ATAM is a refined and improved version of SAAM that helps you review architectural decisions with respect to the quality attributes requirements, and how well they satisfy particular quality goals.
- **Active Design Review (ADR)**
  - ADR is best suited for incomplete or in-progress architectures. The main difference is that the review is more focused on a set of issues or individual sections of the architecture at a time, rather than performing a general review.
- **Active Reviews of Intermediate Designs (ARID)**
  - ARID combines the ADR aspect of reviewing in-progress architecture with a focus on a set of issues, and the ATAM and SAAM approach of scenario-based review focused on quality attributes.
- **Cost Benefit Analysis Method (CBAM)**
  - This CBAM focuses on analyzing the costs, benefits, and schedule implications of architectural decisions.
- **Architecture Level Modifiability Analysis (ALMA)**
  - ALMA evaluates the modifiability of architecture for business information systems (BIS).
- **Family Architecture Assessment Method (FAAM)**
  - FAAM evaluates information system family architectures for interoperability and extensibility.

Source: Microsoft Application Architecture Guide, 2nd Edition (Chapters 1-4)

# ATAM Methods: Presentation

- **1. Present the ATAM.** The method is described to the assembled stakeholders (typically customer representatives, the architect or architecture team, user representatives, maintainers, administrators, managers, testers, integrators, etc.).
- **2. Present business drivers.** The project manager describes what business goals are motivating the development effort and hence what will be the primary architectural drivers (e.g., high availability or time to market or high security).
- **3. Present the architecture.** The architect will describe the proposed architecture, focusing on how it addresses the business drivers.

Source: Carnegie Mellon Software Engineering Institute www.sei.org

# ATAM Methods: Investigation and Analysis

- **4. Identify architectural approaches.** Architectural approaches are identified by the architect, but are not analyzed.
- **5. Generate quality attribute utility tree.** The quality factors that comprise system “utility” (performance, availability, security, modifiability, etc.) are elicited, specified down to the level of scenarios, annotated with stimuli and responses, and prioritized.
- **6. Analyze architectural approaches.** Based upon the high-priority factors identified in Step 5, the architectural approaches that address those factors are elicited and analyzed (for example, an architectural approach aimed at meeting performance goals will be subjected to a performance analysis). During this step architectural risks, sensitivity points, and tradeoff points are identified.

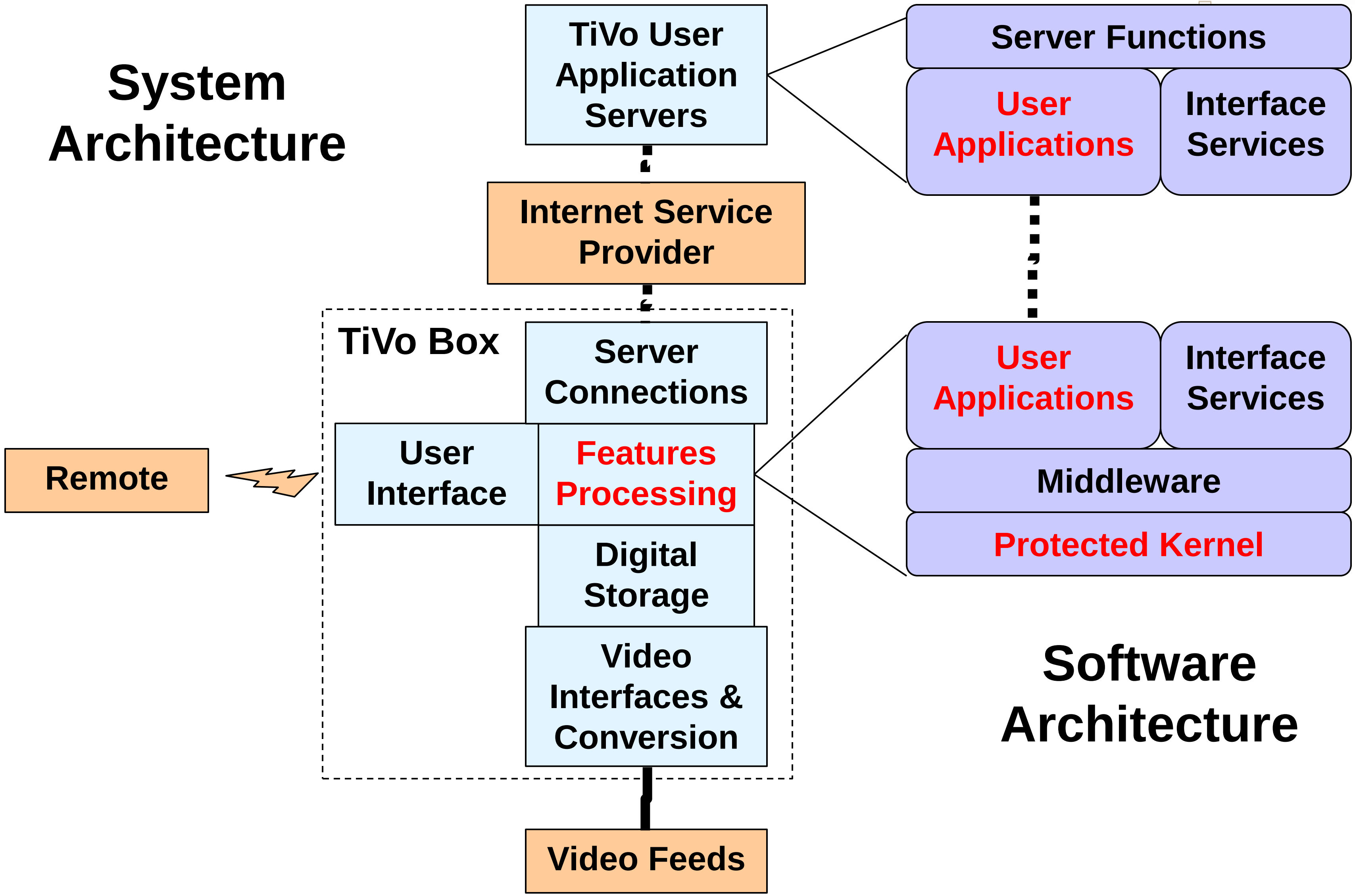
Source: Carnegie Mellon Software Engineering Institute www.sei.org

# ATAM Methods: Testing and Results

- **7. Brainstorm and prioritize scenarios.** Based upon the exemplar scenarios generated in the utility tree step, a larger set of scenarios is elicited from the entire group of stakeholders. This set of scenarios is prioritized via a voting process involving the entire stakeholder group.
- **8. Analyze architectural approaches.** This step reiterates step 6, but here the highly ranked scenarios from Step 7 are considered to be test cases for the analysis of the architectural approaches determined thus far. These test case scenarios may uncover additional architectural approaches, risks, sensitivity points, and tradeoff points which are then documented.
- **9. Present results.** Based upon the information collected in the ATAM (styles, scenarios, attribute-specific questions, the utility tree, risks, sensitivity points, tradeoffs) the ATAM team presents the findings to the assembled stakeholders and potentially writes a report detailing this information along with any proposed mitigation strategies.

Source: Carnegie Mellon Software Engineering Institute www.sei.org

# TiVo Architecture Example



# The Role of the System Architect

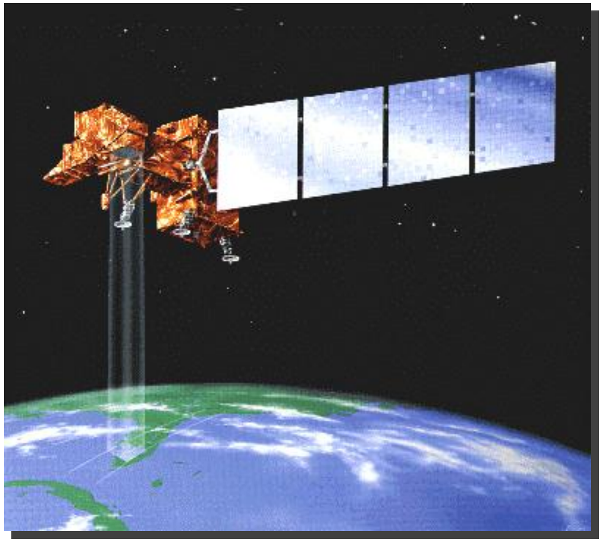
- The System Architect is **more a leadership and management role** than a technical role
- Architects **need experience**, and a blend of management and leadership disciplines
- Communication and vision require leadership capacity
  - The architect holds the architectural vision, often their own
  - The architect makes high-level design decisions around interfaces, functional partitioning, and interactions
  - The architect must communicate these effectively, often visually
- The architect's **primary tasks are rule-setting**
  - The architect must direct technical standards, including design standards, tools, or platforms,
  - These should be based on business goals rather than to place arbitrary restrictions on the choices of developers.

# Leadership Competencies

- **Experience and judgment**
  - The architect must balance the customer's view of the system with their organization's business view of the system
- **Communications**
  - The architecture is presented in visuals to all stakeholders
  - The architecture is derived to written guidelines and design rules for the team
- **Leadership and Systems Thinking**
  - The architecture is the high level vision of the system
  - The architecture is defined more by heuristics than requirements
  - The architecture definition contains a number of soft requirements that have to be evaluated in collaborative groups
- **Management**
  - The architect ensures the design team follows design standards

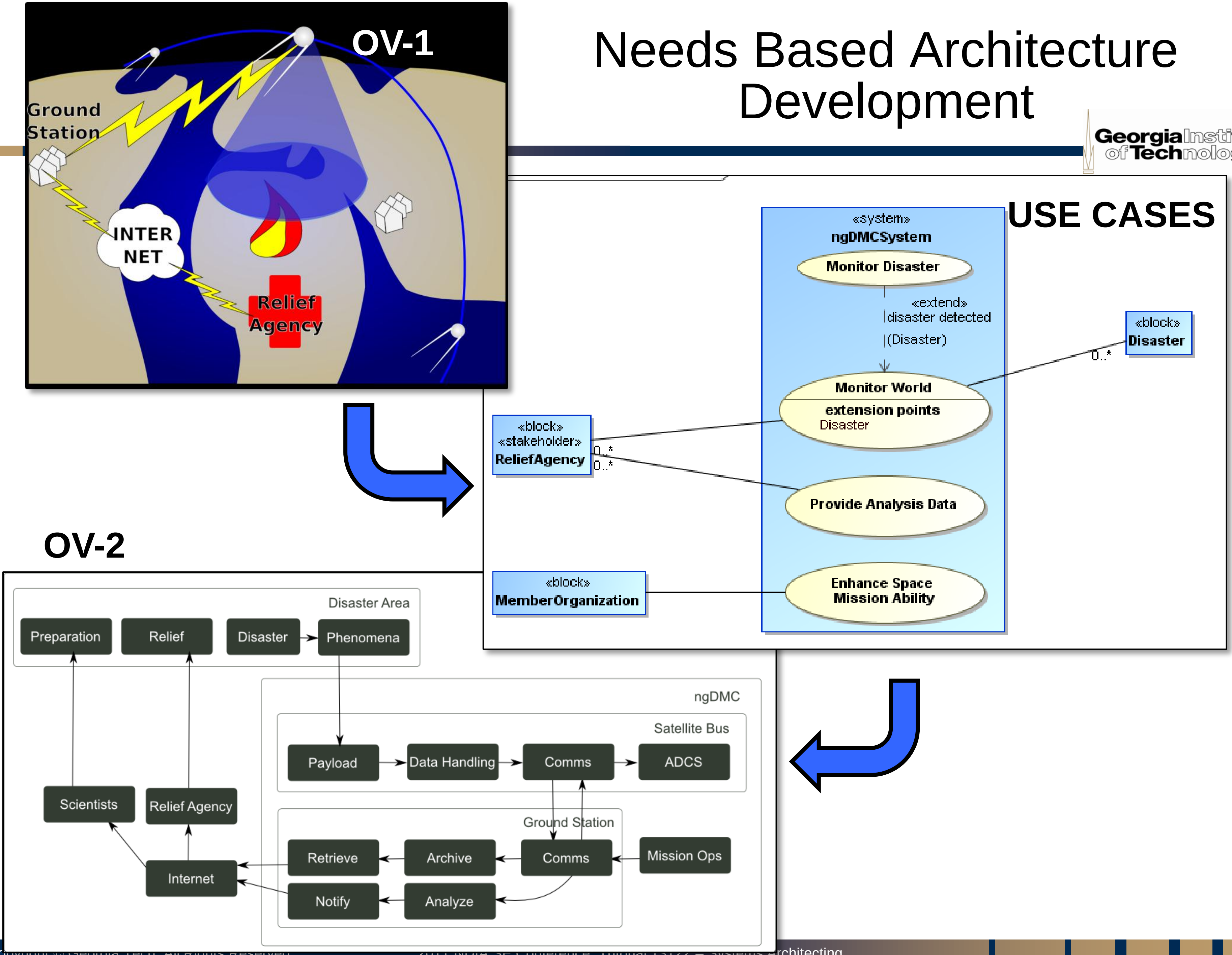
# Architecting Case Study:

# Next Generation Disaster Monitoring Constellation (NGDMC)

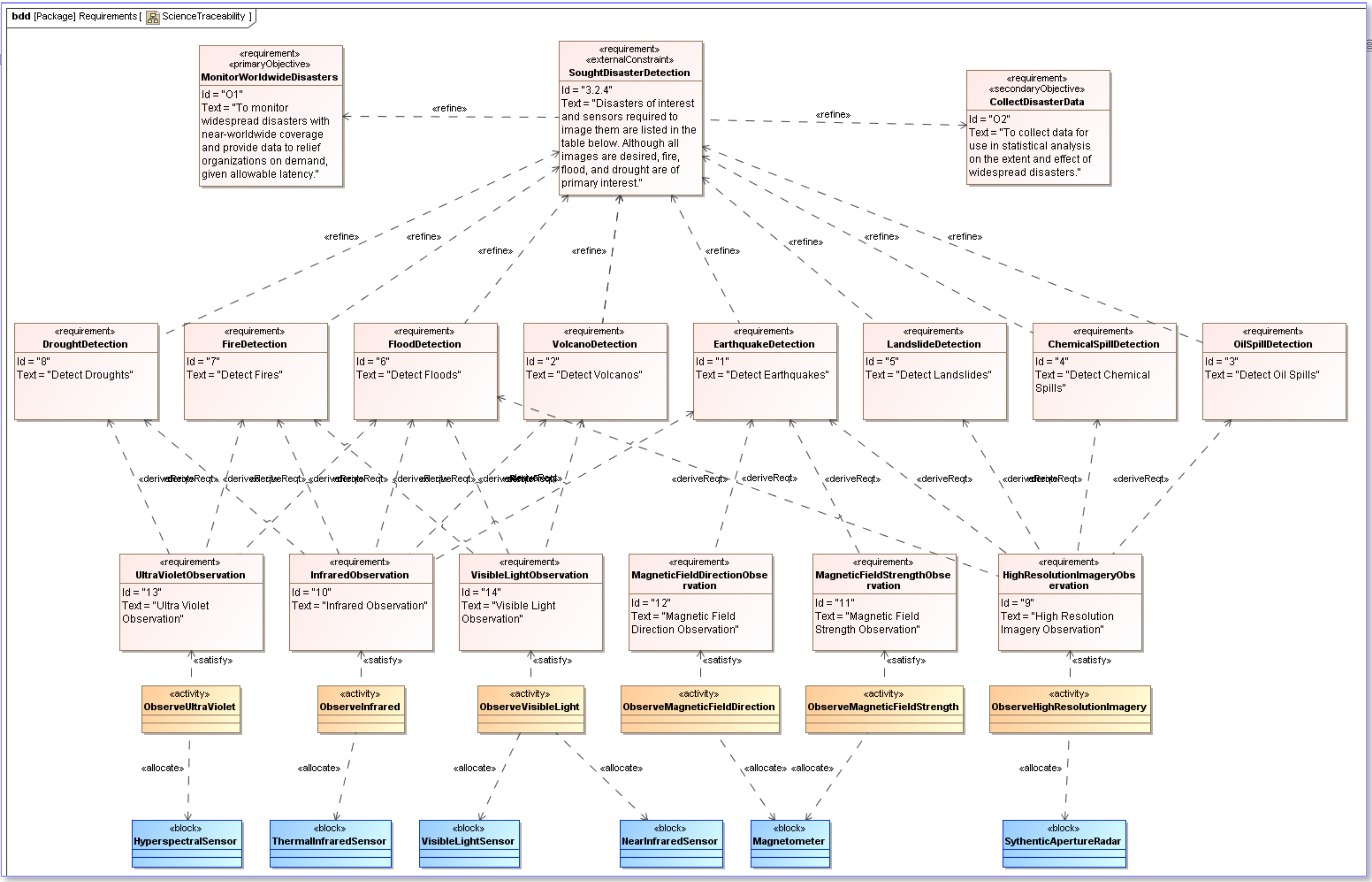


Source: Bollweg, N., Simonetta, L., Pihera, L.D., and King, S., “Systems Engineering Management Plan: Next Generation Disaster Monitoring Constellation,” ASE 6006 Systems Engineering Lab, Fall 2010, Georgia Institute of Technology.

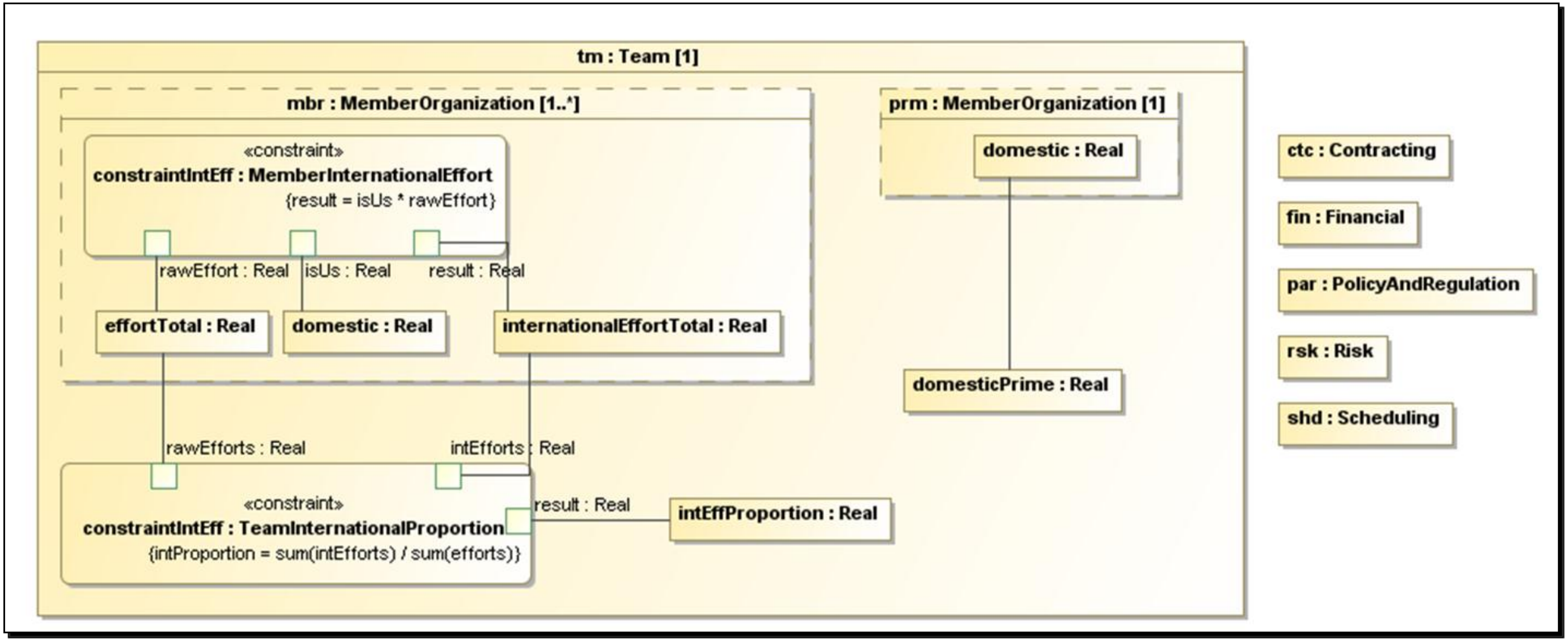
# Needs Based Architecture Development



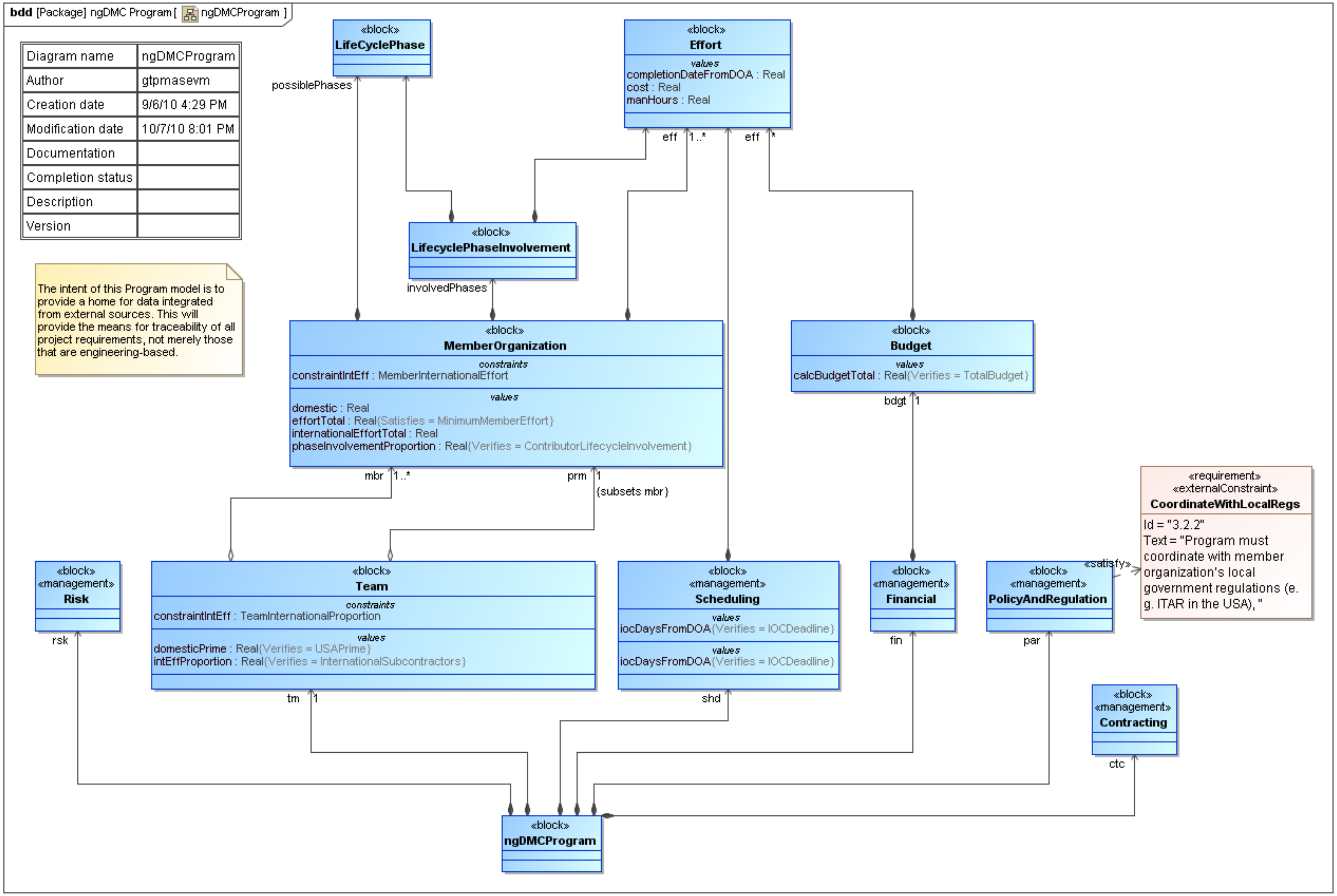
# Requirements Traceability to Architecture



# Programmatic Constraints



# Programmatic Overview



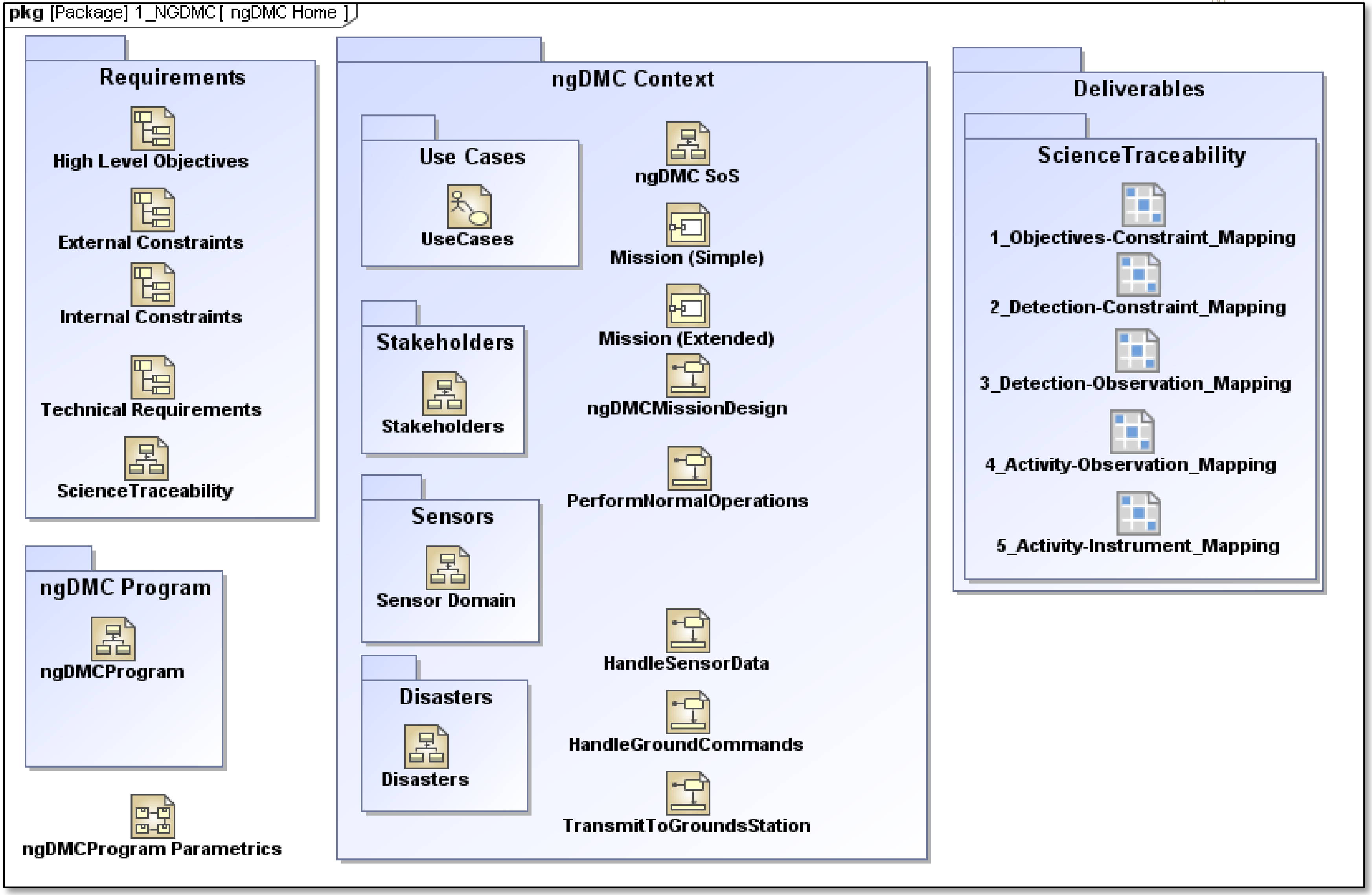
bdd [Package] ngDMC Program [ ngDMCProgram ]

|  |  |
|---|---|
| Diagram name | ngDMCProgram |
| Author | gtpmasevm |
| Creation date | 9/6/10 4:29 PM |
| Modification date | 10/7/10 8:01 PM |
| Documentation |  |
| Completion status |  |
| Description |  |
| Version |  |

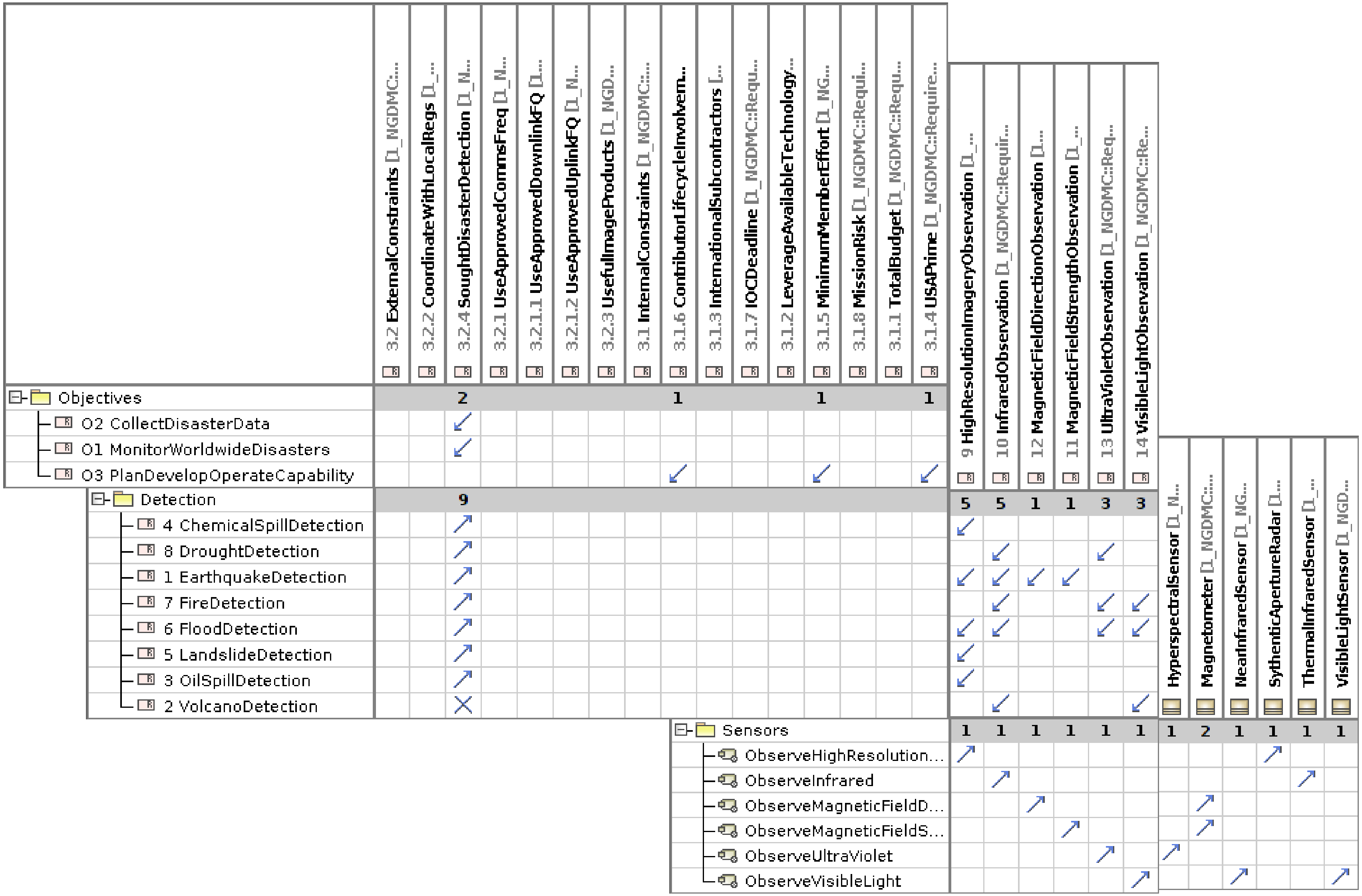
The intent of this Program model is to provide a home for data integrated from external sources. This will provide the means for traceability of all project requirements, not merely those that are engineering-based.

«requirement» «externalConstraint» **CoordinateWithLocalRegs**
Id = "3.2.2"
Text = "Program must coordinate with member organization's local government regulations (e. g. ITAR in the USA), "

# What is the architect’s view here?



# Science and Instruments Traceability Matrix

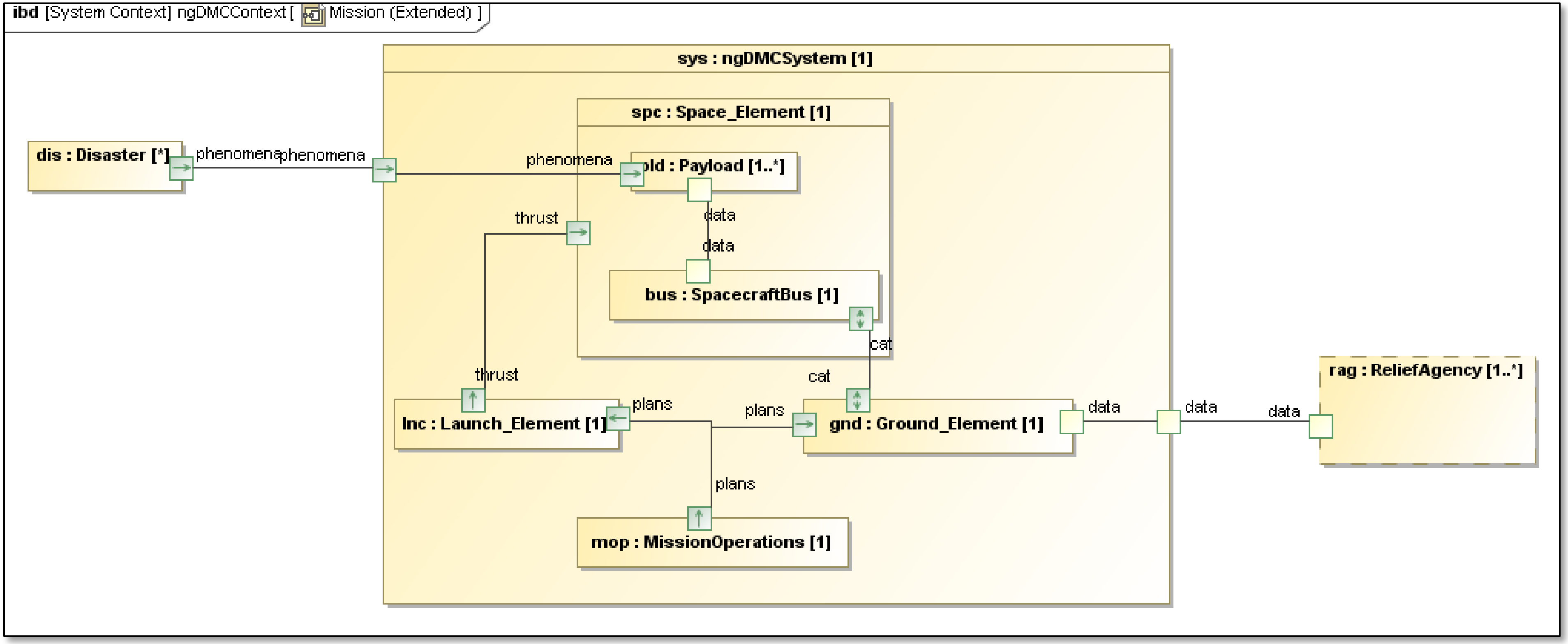


| | 3.2 ExternalConstraints [1_NGDMC:... | 3.2.2 CoordinateWithLocalRegs [1_... | 3.2.4 SoughtDisasterDetection [1_N... | 3.2.1 UseApprovedCommsFreq [1_N... | 3.2.1.1 UseApprovedDownlinkFQ [1... | 3.2.1.2 UseApprovedUplinkFQ [1_N... | 3.2.3 UsefullImageProducts [1_NGD... | 3.1 InternalConstraints [1_NGDMC:... | 3.1.6 ContributorLifecycleInvolvem... | 3.1.3 InternationalSubcontractors [... | 3.1.7 IOCDeadline [1_NGDMC::Requ... | 3.1.2 LeverageAvailableTechnology... | 3.1.5 MinimumMemberEffort [1_NG... | 3.1.8 MissionRisk [1_NGDMC::Requi... | 3.1.1 TotalBudget [1_NGDMC::Requ... | 3.1.4 USAPrime [1_NGDMC::Require... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ⊟ Objectives | | | 2 | | | | | | 1 | | | | 1 | | | 1 |
| O2 CollectDisasterData | | | ↙ | | | | | | | | | | | | | |
| O1 MonitorWorldwideDisasters | | | ↙ | | | | | | | | | | | | | |
| O3 PlanDevelopOperateCapability | | | | | | | | | ↙ | | | | ↙ | | | ↙ |
| ⊟ Detection | | | 9 | | | | | | | | | | | | | |
| 4 ChemicalSpillDetection | | | ↗ | | | | | | | | | | | | | |
| 8 DroughtDetection | | | ↗ | | | | | | | | | | | | | |
| 1 EarthquakeDetection | | | ↗ | | | | | | | | | | | | | |
| 7 FireDetection | | | ↗ | | | | | | | | | | | | | |
| 6 FloodDetection | | | ↗ | | | | | | | | | | | | | |
| 5 LandslideDetection | | | ↗ | | | | | | | | | | | | | |
| 3 OilSpillDetection | | | ↗ | | | | | | | | | | | | | |
| 2 VolcanoDetection | | | ✕ | | | | | | | | | | | | | |

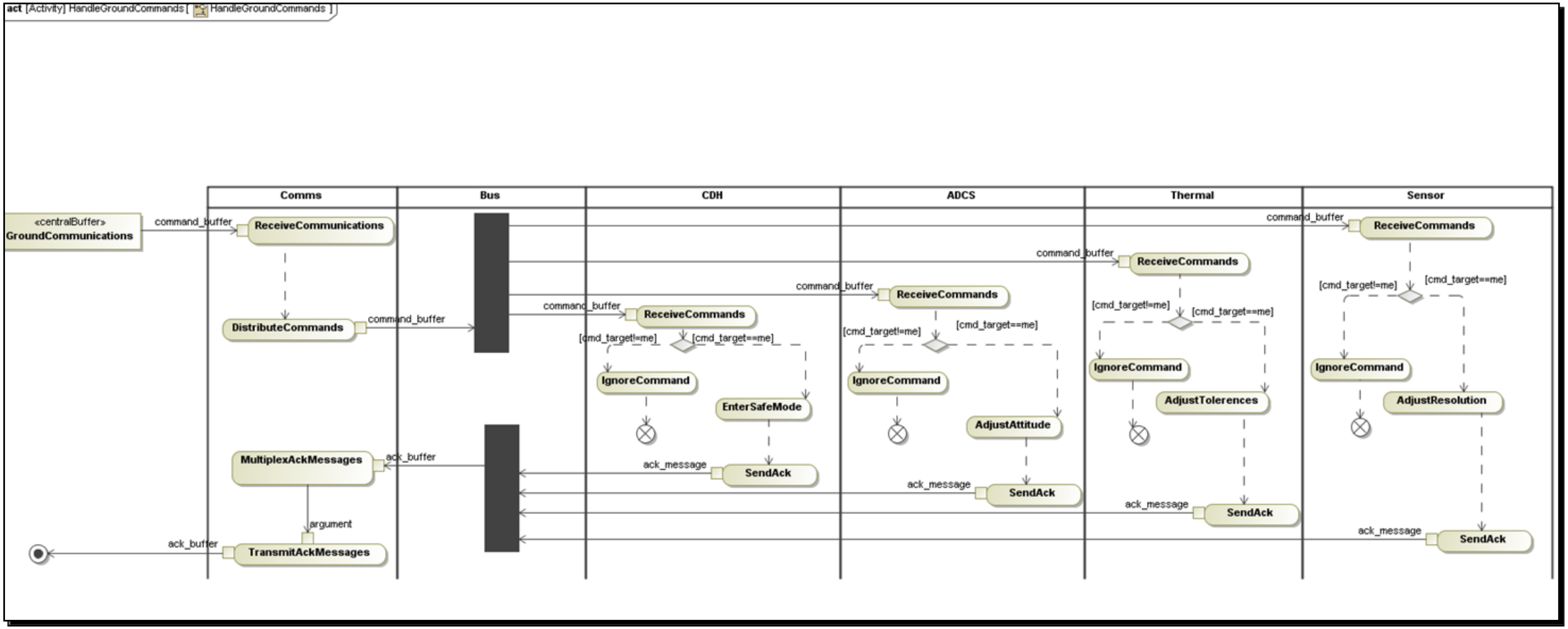
| | 9 HighResolutionImageryObservation [1_... | 10 InfraredObservation [1_NGDMC::Requir... | 12 MagneticFieldDirectionObservation [1... | 11 MagneticFieldStrengthObservation [1_... | 13 UltraVioletObservation [1_NGDMC::Req... | 14 VisibleLightObservation [1_NGDMC::Re... |
|---|---|---|---|---|---|---|
| ⊟ Detection | 5 | 5 | 1 | 1 | 3 | 3 |
| 4 ChemicalSpillDetection | ↙ | | | | | |
| 8 DroughtDetection | | ↙ | | | ↙ | |
| 1 EarthquakeDetection | ↙ | ↙ | ↙ | ↙ | | |
| 7 FireDetection | | ↙ | | | ↙ | ↙ |
| 6 FloodDetection | ↙ | ↙ | | | ↙ | ↙ |
| 5 LandslideDetection | ↙ | | | | | |
| 3 OilSpillDetection | ↙ | | | | | |
| 2 VolcanoDetection | | ↙ | | | | ↙ |
| ⊟ Sensors | 1 | 1 | 1 | 1 | 1 | 1 |
| ObserveHighResolution... | ↗ | | | | | |
| ObserveInfrared | | ↗ | | | | |
| ObserveMagneticFieldD... | | | ↗ | | | |
| ObserveMagneticFieldS... | | | | ↗ | | |
| ObserveUltraViolet | | | | | ↗ | |
| ObserveVisibleLight | | | | | | ↗ |

| | HyperspectralSensor [1_N... | Magnetometer [1_NGDMC:... | NearInfraredSensor [1_NG... | SythenticApertureRadar [1... | ThermalInfraredSensor [1_... | VisibleLightSensor [1_NGD... |
|---|---|---|---|---|---|---|
| ⊟ Sensors | 1 | 2 | 1 | 1 | 1 | 1 |
| ObserveHighResolution... | | | | ↗ | | |
| ObserveInfrared | | | | | ↗ | |
| ObserveMagneticFieldD... | | ↗ | | | | |
| ObserveMagneticFieldS... | | ↗ | | | | |
| ObserveUltraViolet | ↗ | | | | | |
| ObserveVisibleLight | | | ↗ | | | ↗ |

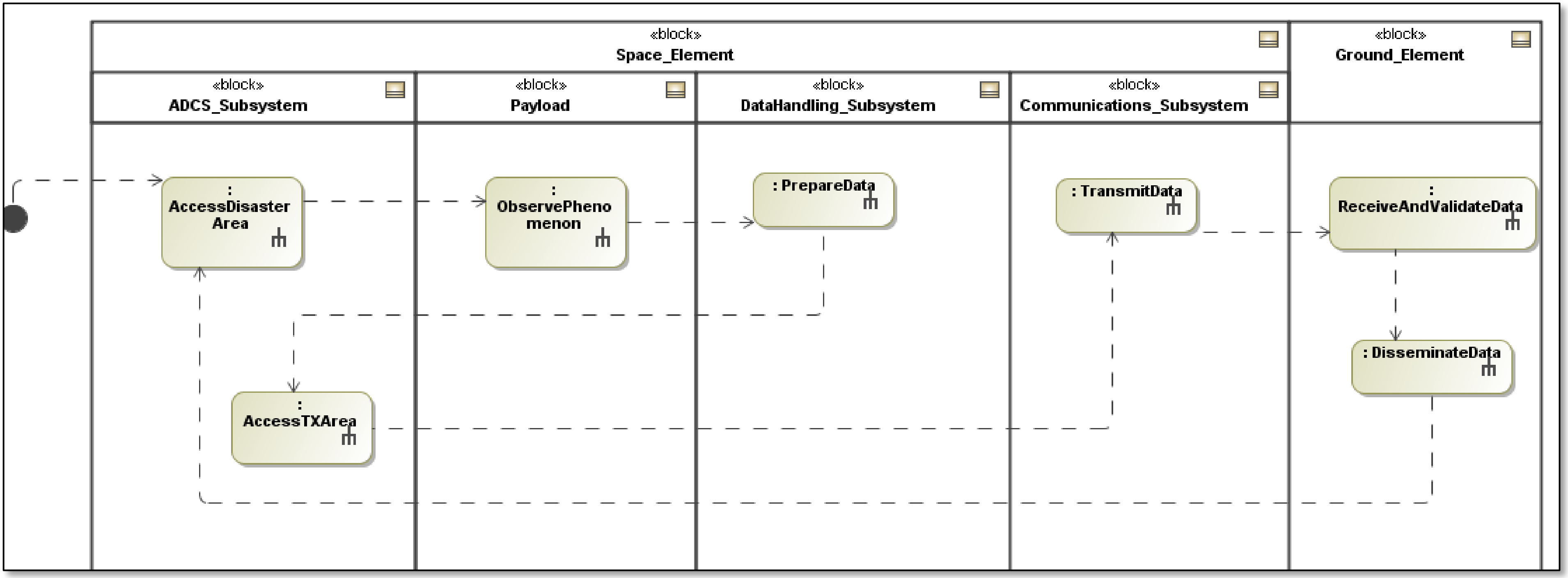
# Mission Architecture



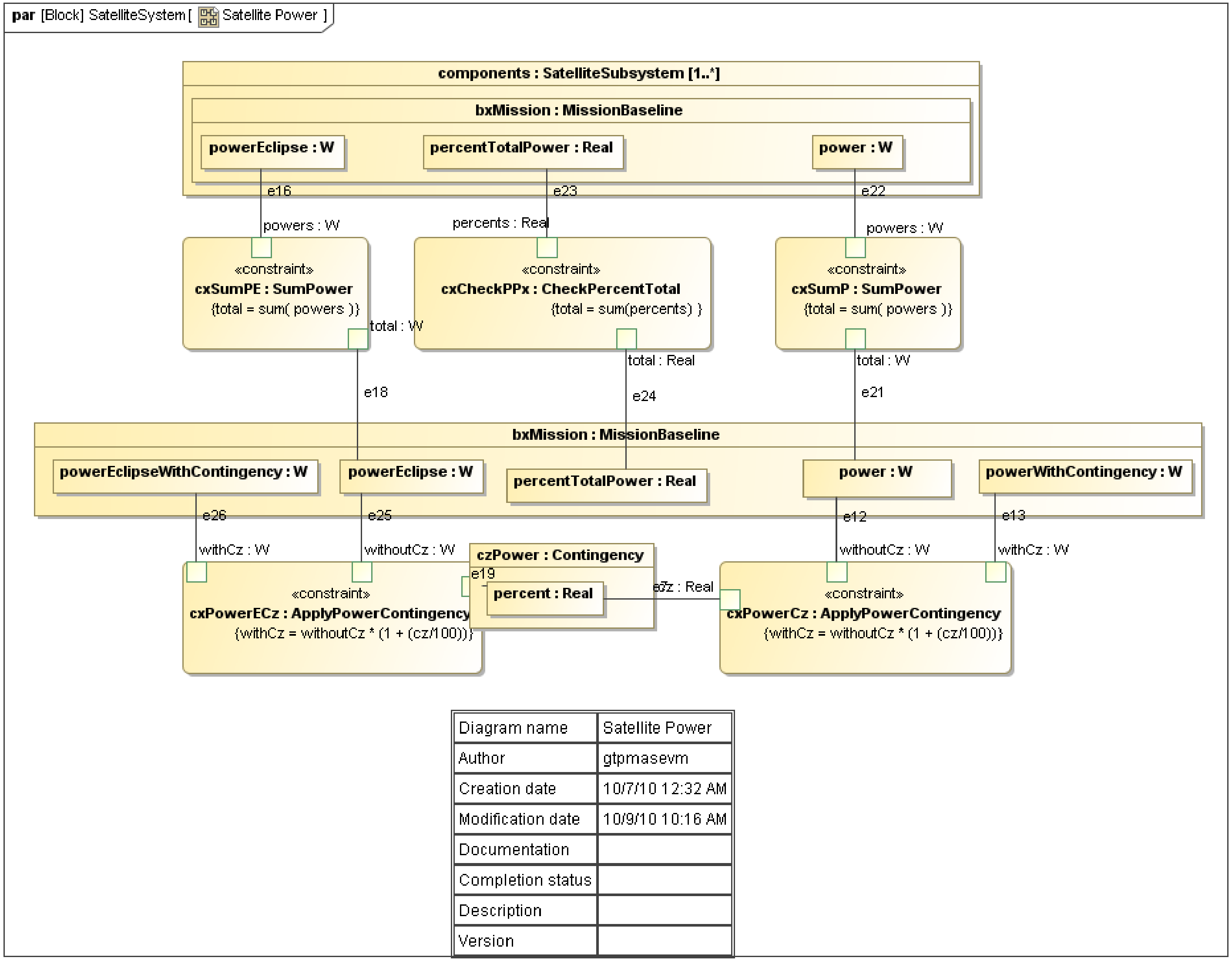
# Functional Flow



# Functional Flow

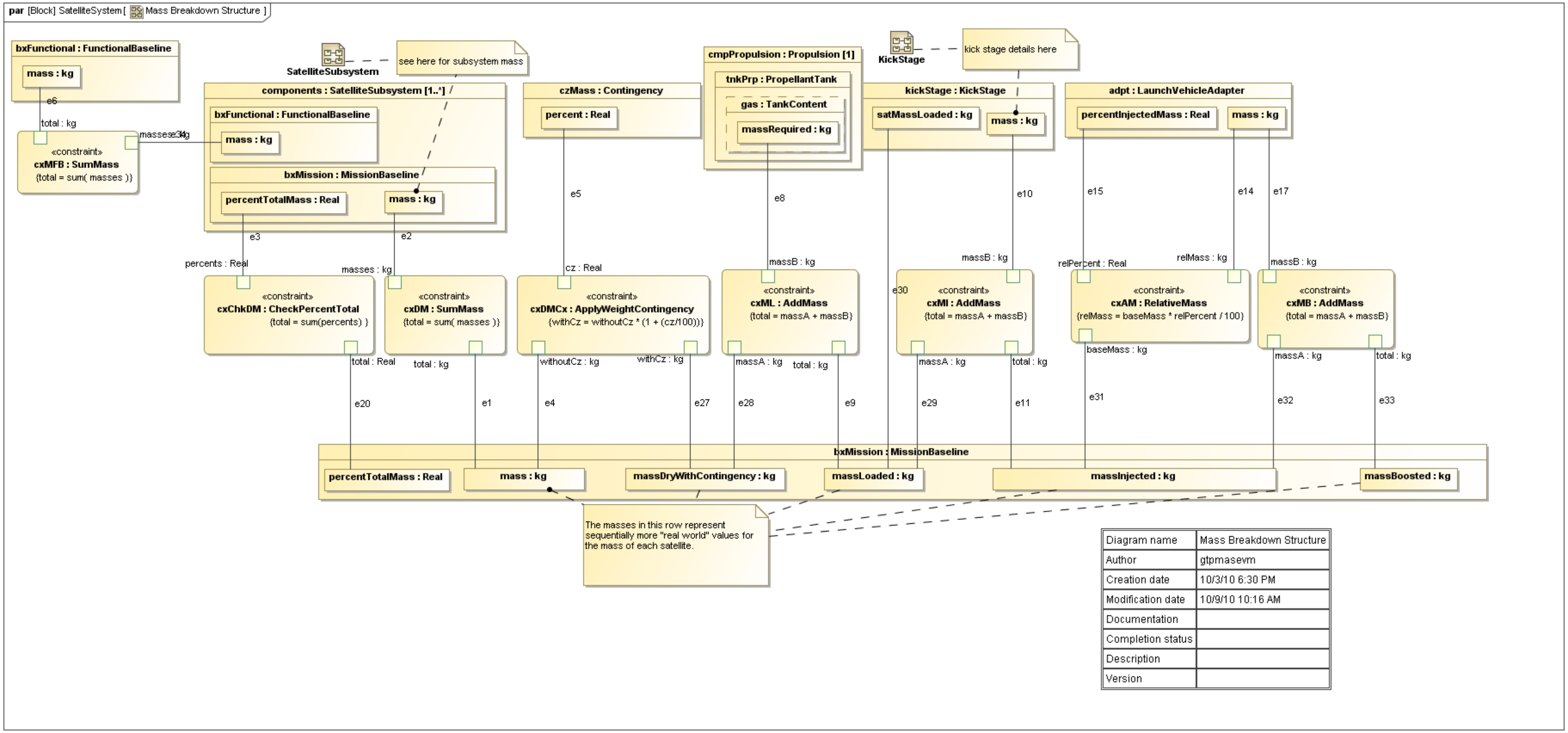


# Constraints



| | |
|---|---|
| Diagram name | Satellite Power |
| Author | gtpmasevm |
| Creation date | 10/7/10 12:32 AM |
| Modification date | 10/9/10 10:16 AM |
| Documentation | |
| Completion status | |
| Description | |
| Version | |

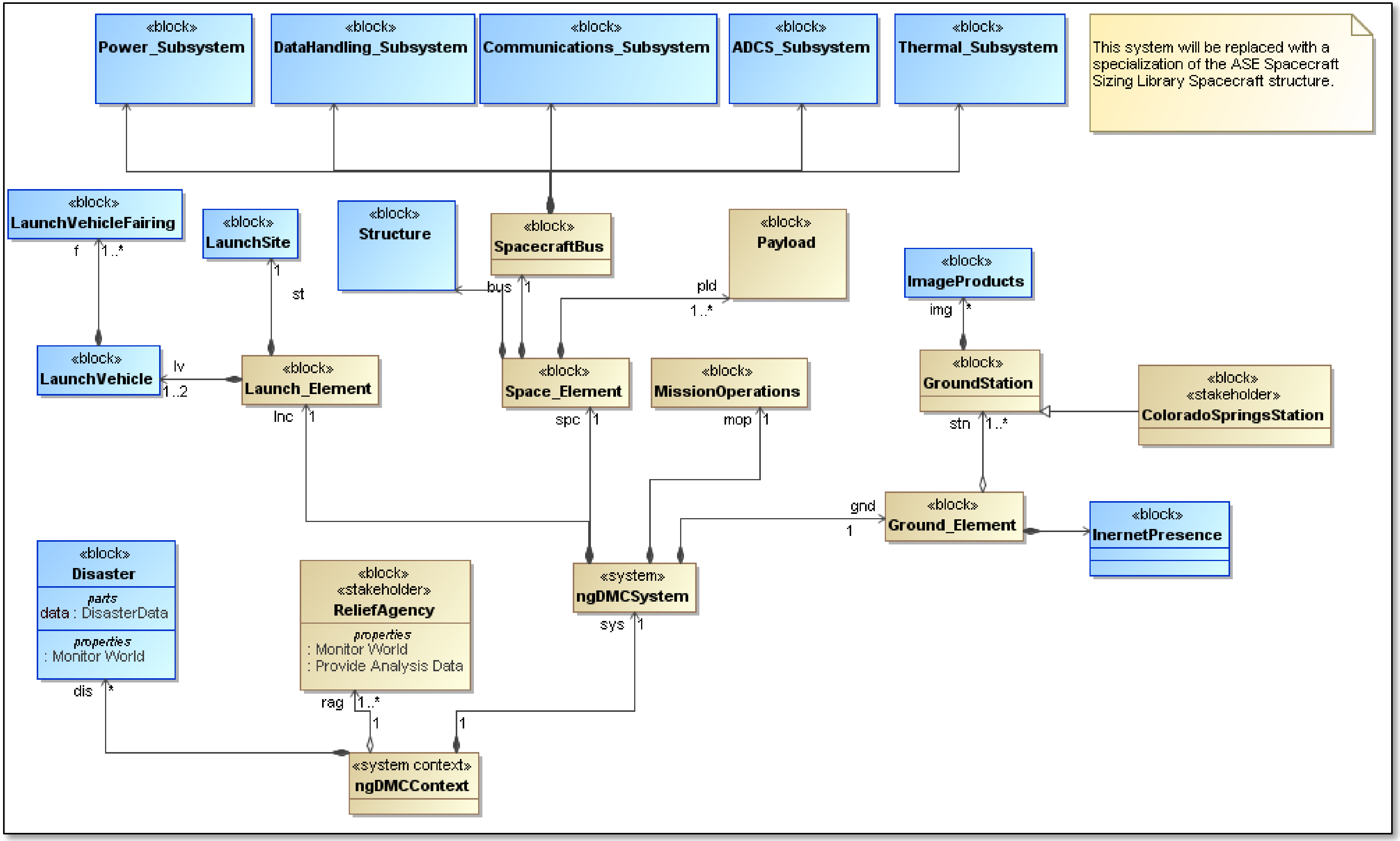
# Constraints



The masses in this row represent sequentially more "real world" values for the mass of each satellite.

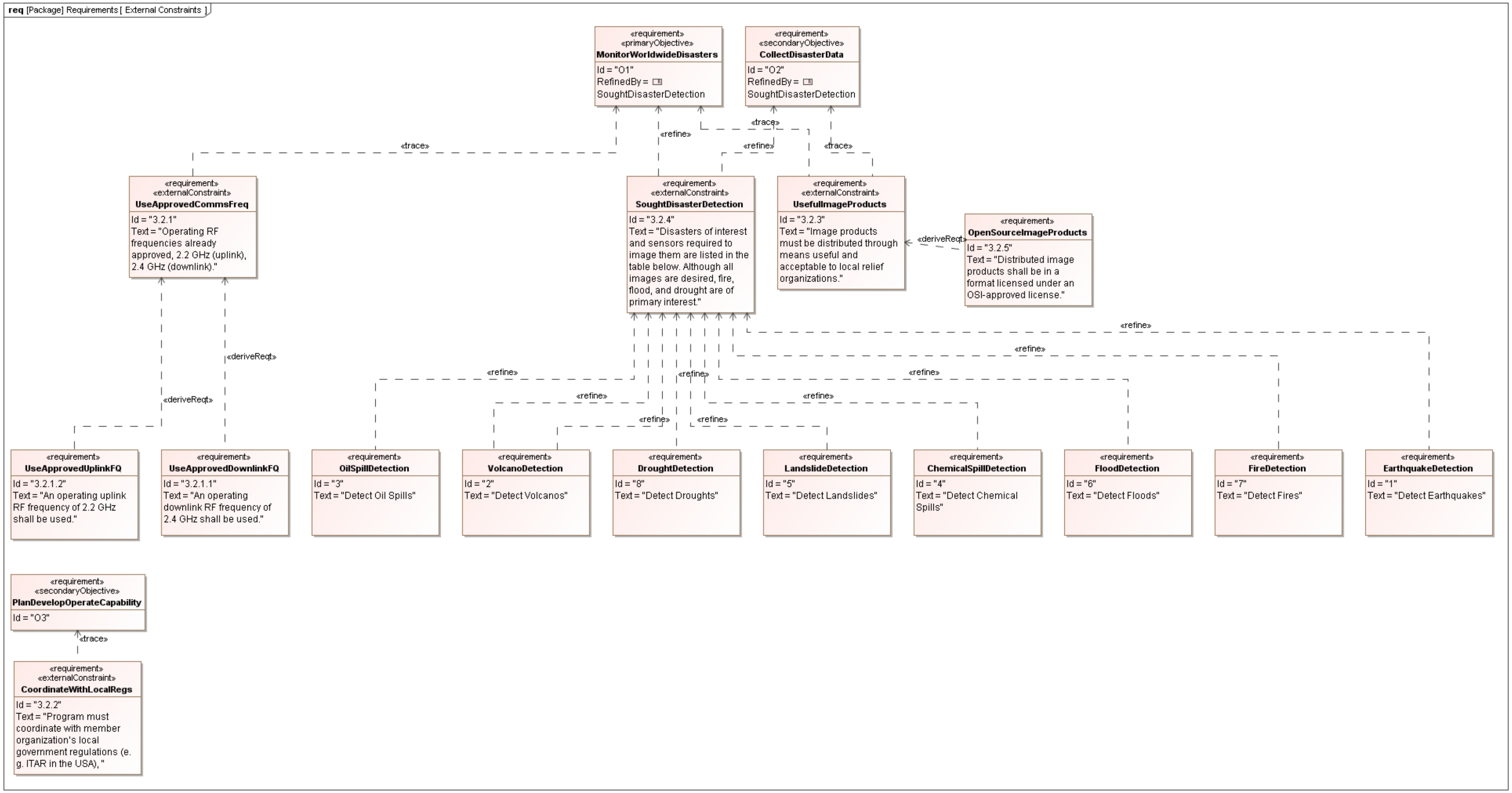
| | |
|---|---|
| Diagram name | Mass Breakdown Structure |
| Author | gtpmasevm |
| Creation date | 10/3/10 6:30 PM |
| Modification date | 10/9/10 10:16 AM |
| Documentation | |
| Completion status | |
| Description | |
| Version | |

# SoS / subsystem view



This system will be replaced with a specialization of the ASE Spacecraft Sizing Library Spacecraft structure.

# External Constraints


**req** [Package] Requirements [ External Constraints ]

«requirement»
«primaryObjective»
**MonitorWorldwideDisasters**
Id = "O1"
RefinedBy = SoughtDisasterDetection

«requirement»
«secondaryObjective»
**CollectDisasterData**
Id = "O2"
RefinedBy = SoughtDisasterDetection

«requirement»
«externalConstraint»
**UseApprovedCommsFreq**
Id = "3.2.1"
Text = "Operating RF frequencies already approved, 2.2 GHz (uplink), 2.4 GHz (downlink)."

«requirement»
«externalConstraint»
**SoughtDisasterDetection**
Id = "3.2.4"
Text = "Disasters of interest and sensors required to image them are listed in the table below. Although all images are desired, fire, flood, and drought are of primary interest."

«requirement»
«externalConstraint»
**UsefulImageProducts**
Id = "3.2.3"
Text = "Image products must be distributed through means useful and acceptable to local relief organizations."

«requirement»
**OpenSourceImageProducts**
Id = "3.2.5"
Text = "Distributed image products shall be in a format licensed under an OSI-approved license."

«requirement»
**UseApprovedUplinkFQ**
Id = "3.2.1.2"
Text = "An operating uplink RF frequency of 2.2 GHz shall be used."

«requirement»
**UseApprovedDownlinkFQ**
Id = "3.2.1.1"
Text = "An operating downlink RF frequency of 2.4 GHz shall be used."

«requirement»
**OilSpillDetection**
Id = "3"
Text = "Detect Oil Spills"

«requirement»
**VolcanoDetection**
Id = "2"
Text = "Detect Volcanos"

«requirement»
**DroughtDetection**
Id = "8"
Text = "Detect Droughts"

«requirement»
**LandslideDetection**
Id = "5"
Text = "Detect Landslides"

«requirement»
**ChemicalSpillDetection**
Id = "4"
Text = "Detect Chemical Spills"

«requirement»
**FloodDetection**
Id = "6"
Text = "Detect Floods"

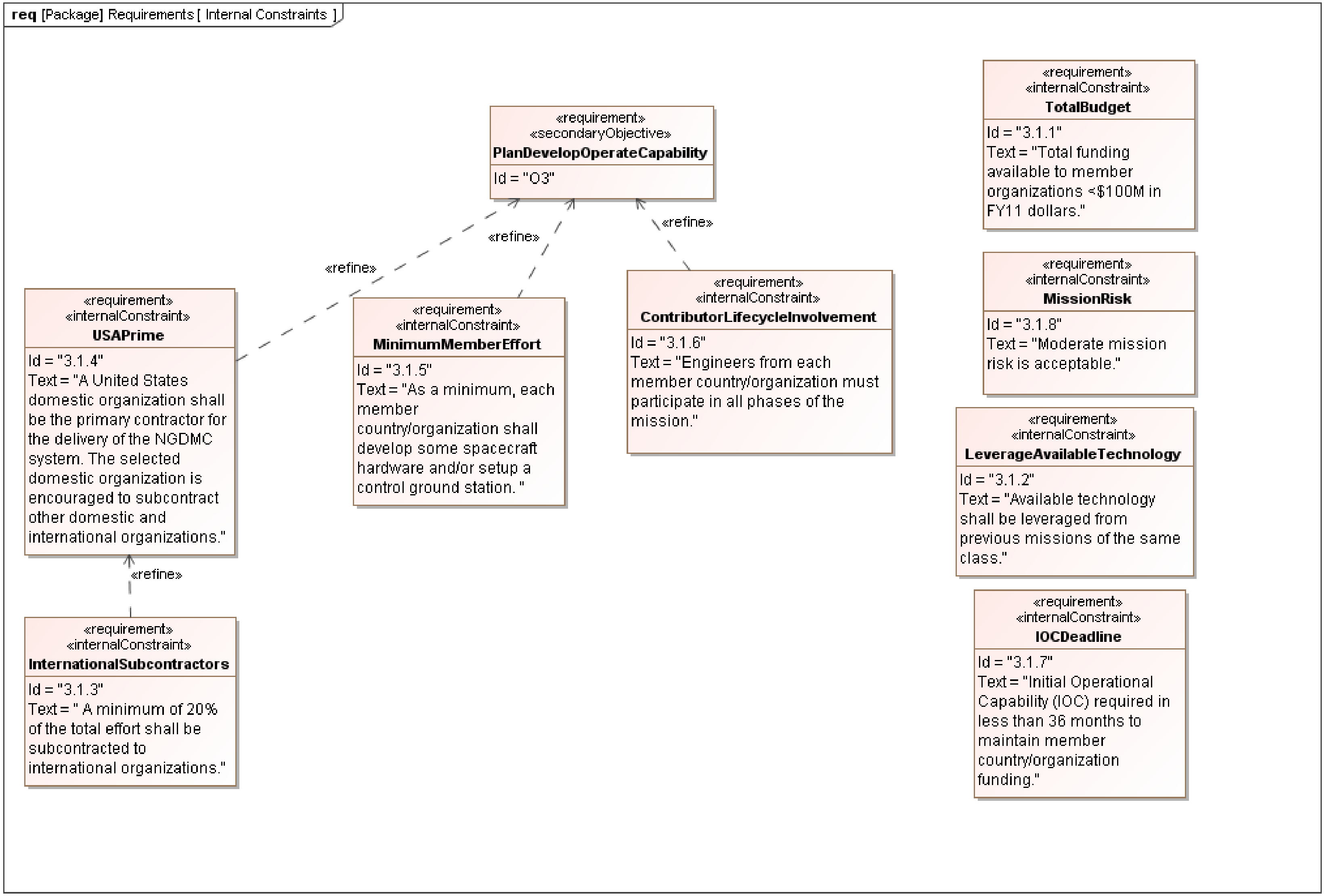
«requirement»
**FireDetection**
Id = "7"
Text = "Detect Fires"

«requirement»
**EarthquakeDetection**
Id = "1"
Text = "Detect Earthquakes"

«requirement»
«secondaryObjective»
**PlanDevelopOperateCapability**
Id = "O3"

«requirement»
«externalConstraint»
**CoordinateWithLocalRegs**
Id = "3.2.2"
Text = "Program must coordinate with member organization's local government regulations (e.g. ITAR in the USA),"

# Internal Constraints



**req** [Package] Requirements [ Internal Constraints ]

«requirement»
«secondaryObjective»
**PlanDevelopOperateCapability**
Id = "O3"

«requirement»
«internalConstraint»
**TotalBudget**
Id = "3.1.1"
Text = "Total funding available to member organizations <$100M in FY11 dollars."

«requirement»
«internalConstraint»
**USAPrime**
Id = "3.1.4"
Text = "A United States domestic organization shall be the primary contractor for the delivery of the NGDMC system. The selected domestic organization is encouraged to subcontract other domestic and international organizations."

«requirement»
«internalConstraint»
**MinimumMemberEffort**
Id = "3.1.5"
Text = "As a minimum, each member country/organization shall develop some spacecraft hardware and/or setup a control ground station. "

«requirement»
«internalConstraint»
**ContributorLifecycleInvolvement**
Id = "3.1.6"
Text = "Engineers from each member country/organization must participate in all phases of the mission."

«requirement»
«internalConstraint»
**MissionRisk**
Id = "3.1.8"
Text = "Moderate mission risk is acceptable."

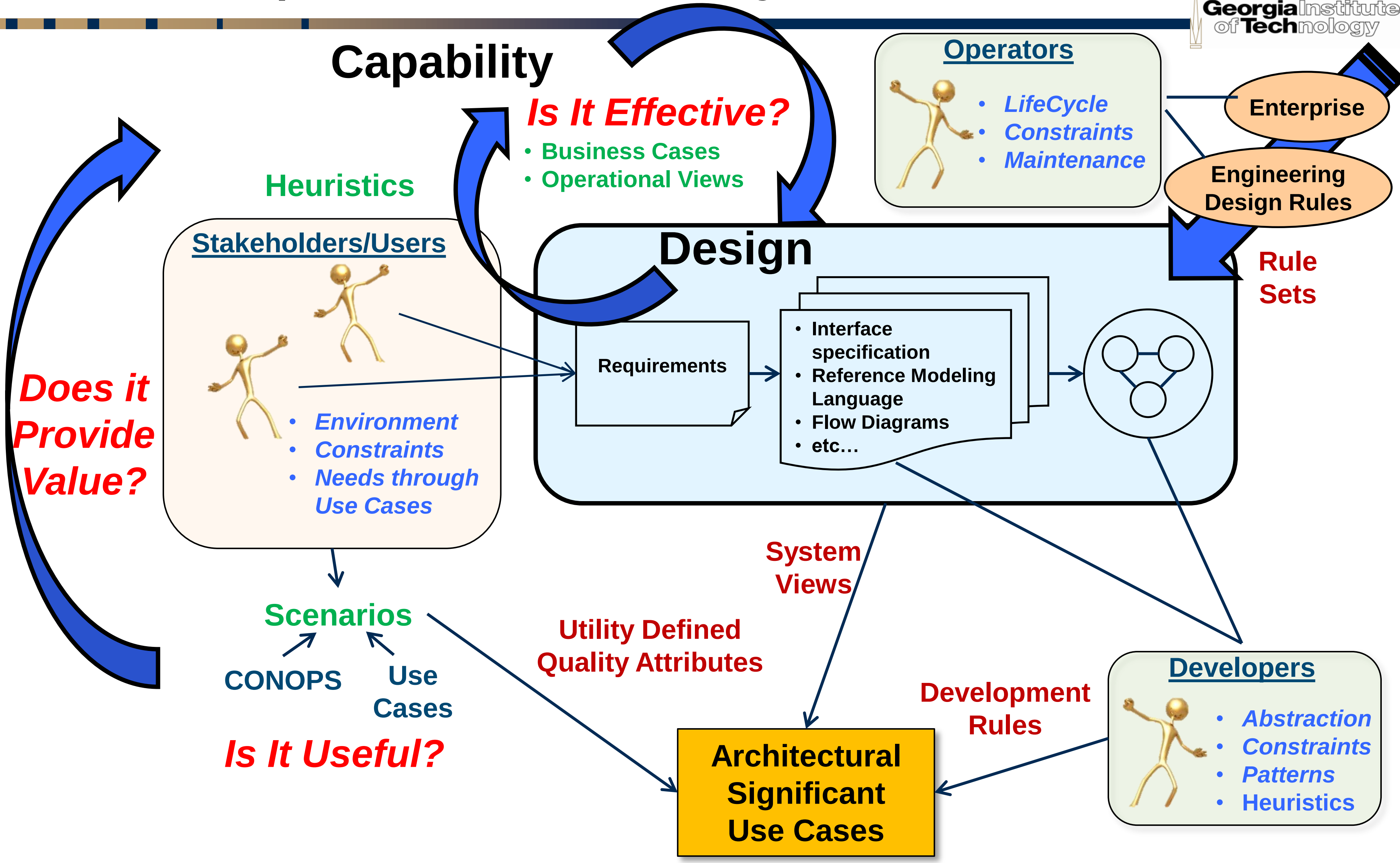
«requirement»
«internalConstraint»
**LeverageAvailableTechnology**
Id = "3.1.2"
Text = "Available technology shall be leveraged from previous missions of the same class."

«requirement»
«internalConstraint»
**InternationalSubcontractors**
Id = "3.1.3"
Text = " A minimum of 20% of the total effort shall be subcontracted to international organizations."

«requirement»
«internalConstraint»
**IOCDeadline**
Id = "3.1.7"
Text = "Initial Operational Capability (IOC) required in less than 36 months to maintain member country/organization funding."

# Conclusions

# Perspective of the Systems Architect



**Capability**

***Is It Effective?***

- **Business Cases**
- **Operational Views**

**Heuristics**

**Stakeholders/Users**

- *Environment*
- *Constraints*
- *Needs through Use Cases*

***Does it Provide Value?***

**Operators**

- *LifeCycle*
- *Constraints*
- *Maintenance*

**Enterprise**

**Engineering Design Rules**

**Rule Sets**

**Design**

**Requirements**

- **Interface specification**
- **Reference Modeling Language**
- **Flow Diagrams**
- **etc…**

**Scenarios**

**CONOPS**

**Use Cases**

***Is It Useful?***

**System Views**

**Utility Defined Quality Attributes**

**Development Rules**

**Developers**

- *Abstraction*
- *Constraints*
- *Patterns*
- **Heuristics**

**Architectural Significant Use Cases**

# Summary and Conclusions

- Classic **systems architecting** provides fundamental representation through views and view points
- Incremental development of ill defined or evolving systems through **agile development**
- **Scenario based methods** for evaluating quality are effective in the context of satisfying business drivers
- Architect serves as a leader on the development team, employing practical **management methods**

# Review of Tutorial Goals

- Introduce the student to **methods** and **practices** for systems architecting
- Apply **agile principles** and incremental development to architecting
- Learn **novel methods** for combining narrative, visual, and specification techniques for rapid and incremental architecture development
- Learn practical approaches to **facilitate the process** introduced in this tutorial

# Primary References*

- Mark W. Maier, The Art of Systems Architecting, Third Edition, CRC Press, 2009.
- Mo Jamshidi (Ed.), Systems of Systems Engineering: Principles and Applications, CRC Press, 2009.
- IEEE-STD-1471-2000, “Systems and software engineering —Recommended practice for architectural description of software-intensive systems”
- Kossiakoff, A. and Sweet, W.N., Systems Engineering Principles and Practice, John Wiley & Sons, 2003.
- Martin, R.C., Agile Software Development, Principles, Patterns, and Practices, Prentice Hall, 2002.

*** Other references used in this tutorial are cited on appropriate slides**