The Role of Models in Systems Engineering

Barclay Brown, ESEP
Global Solution Executive, IBM Rational
barclayb@us.ibm.com

Graham Bleakley Ph.D, Unleash the Labs
IBM Rational
graham.bleakley@uk.ibm.com

IBM Software

Innovate2012

The Premier Event for Software and Systems Innovation

Next NOW!

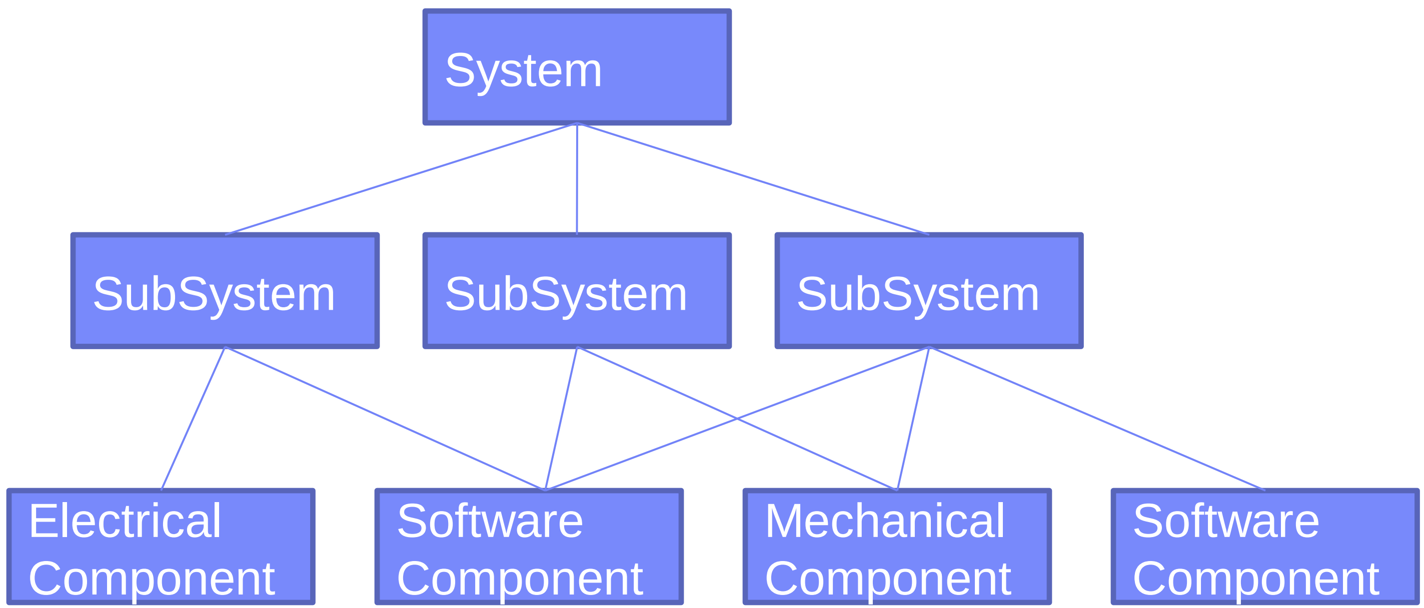
Systems Engineering Addresses Broad Concerns

- What concerns fall squarely into a **specific** engineering discipline?
- What concerns are **independent** of the specific engineering disciplines?

Systems Engineering has both a breadth and depth perspective.

Systems Engineering
Breadth Perspective
Cross-discipline analysis, system-of-systems modeling…

Systems Engineering
Depth Perspective
Requirements engineering, configuration management…



Systems Eng... ering has both a breadth and depth perspective.

Systems Engineering
Breadth Perspective
Cross-discipline analysis, system-o... ms modeling…

Sys... Engineering
D... erspective
...irements engineering,
...iguration management…

System

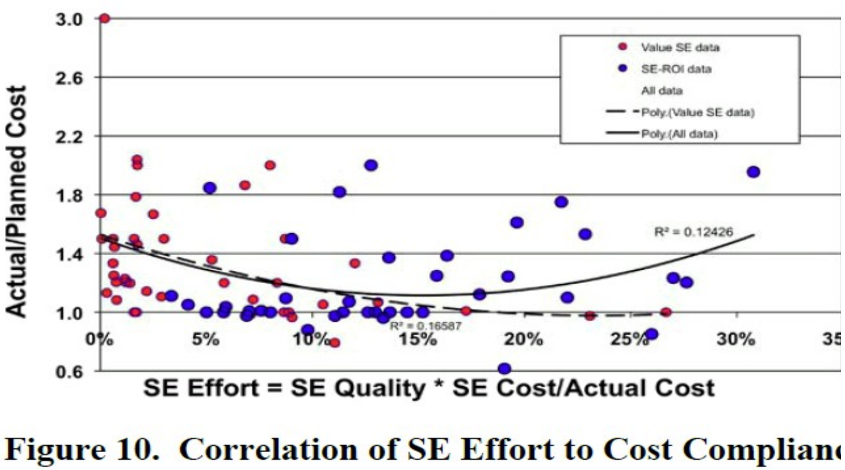
...chanical
...mponent

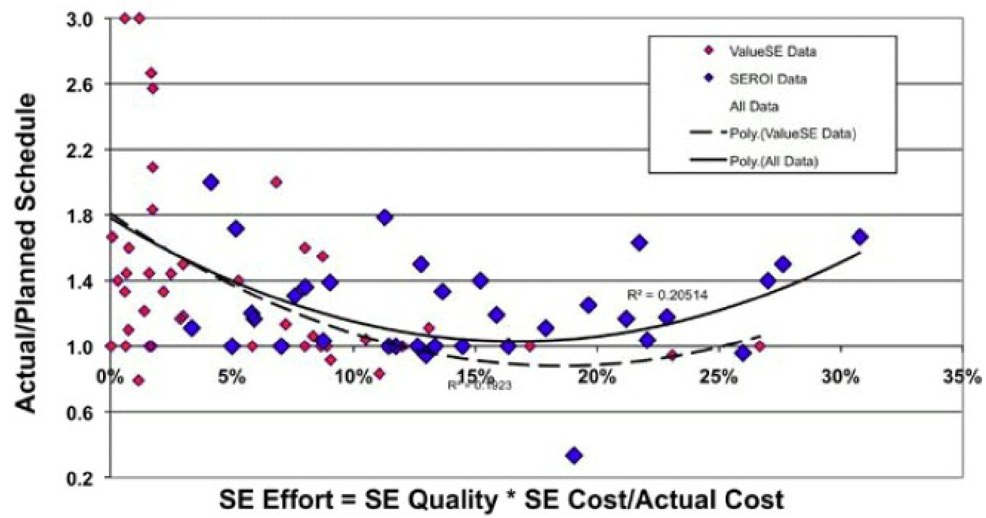
Software
Component

Systems Engineering is a **profession**, a role, even a job title, for those who work exclusively at a ...ystem-of-systems level.

Systems Engineering also names a **set of methods,** skills and techniques that can be applied both at a system-of-systems level and within specific engineering disciplines.

Value of Systems Engineering: Cost and Schedule





Applying the right amount of systems engineering is critical to meeting cost and schedule targets.

Source: Honour, Eric (2010), *Systems Engineering Return on Investment,* University of South Australia, p9

Today's reality: for integrated device and software of systems development

66% — Device software designs completed over budget
EMF 2003

24% — Projects canceled due to unrecoverable slip in schedule
EMF 2003

33% — Produced devices do not meet performance or functionality requirements
EMF 2003

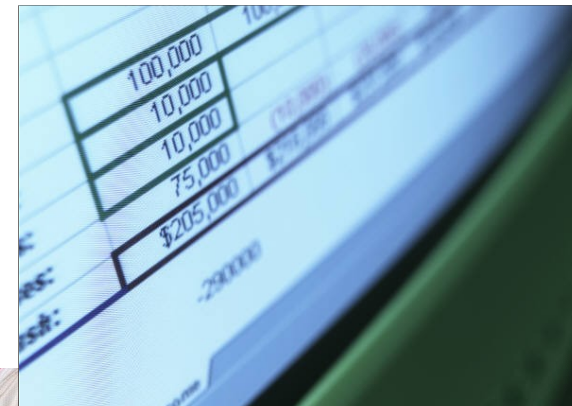
2x — Software content in devices is doubling every two years
IDC 2002

Many notable system failures have been failures in subsystem interfaces, requirements fidelity and system engineering.

- Systems have become more complex through integration
  - E.g. automobiles with multiple ECUs are more like a network of general purpose computers with large network software
- Projects with 700-2000 requirements cannot be held in mind at full detail
  - Models with varied levels of abstraction must be used
- Managing change and understanding impact of change is a $$$ million problem
  - Requirements models and automated tools must be used to be effective

*Interestingly, these are failures of **knowledge** and **communication**, not of engineering.*

What do each of these have to teach us about Advanced Systems Engineering?



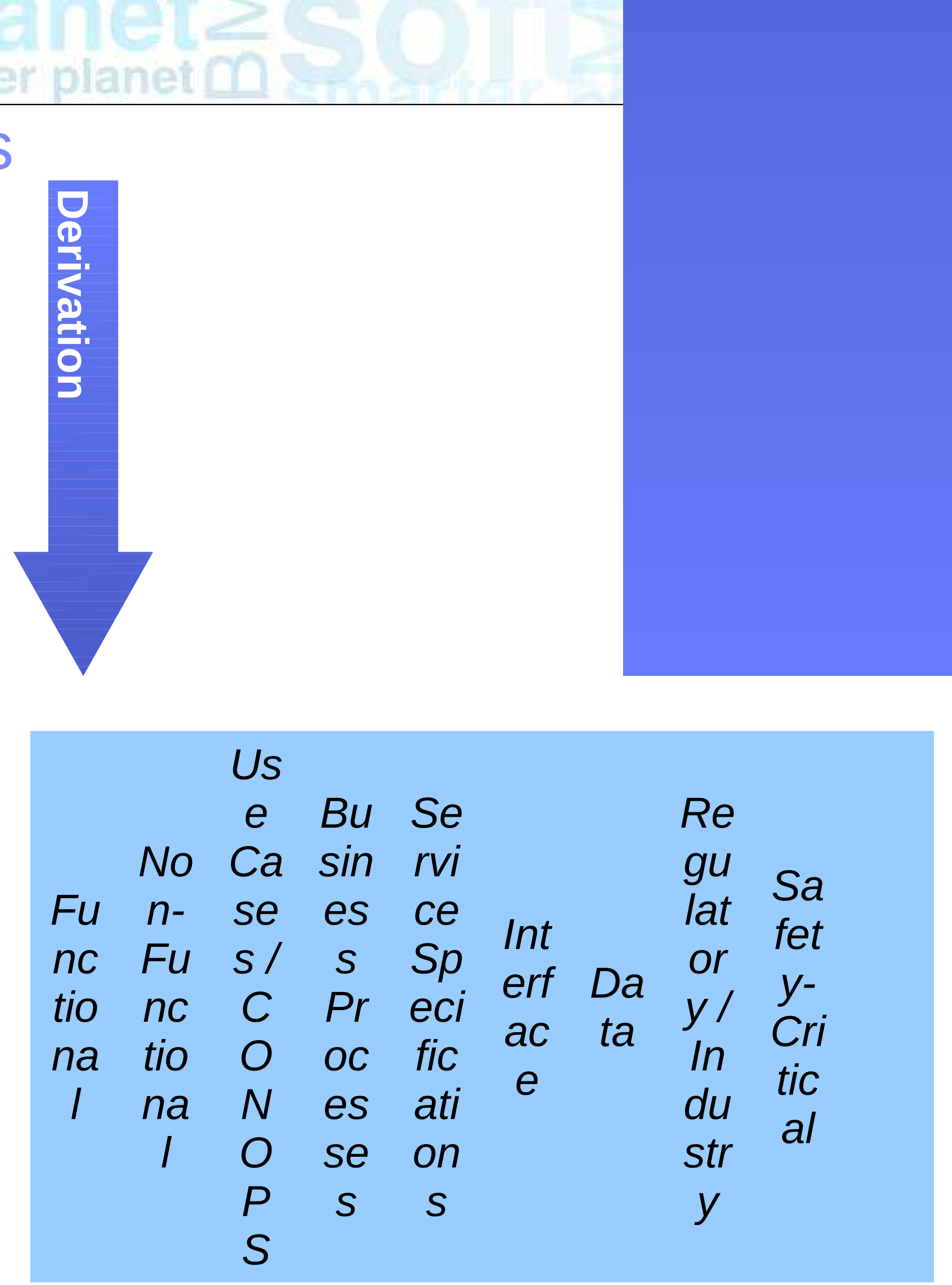
Social Media

Philosophy of Advanced Systems Engineering

- **Models** are better than documents (but documents are needed too)
- Collaborate and **share** information across functions
- Build **Shared Understanding** as hedge against risks
- **Automate** as much as possible
- Insurance: Invest a little now to **avoid large risk** later
- Optimize for **change** (not for stability)
- Be *able* to trace **everything**, but only trace what adds value
- **ABC**: Adopt before Buy, Buy before Create
- **Measure** and improve; closed-loop governance
- Start early: what can we do **NOW**?
- Rewards of doing it right are enormous (**$ Billions**)

Advanced Requirements Management and Modelling

- Requirements are no longer "one kind of thing"
- Recognize levels and types and their relationships
- Implement "live" traceability between all kinds of requirements, including architecture and design
- Link requirements to design, implementation, verification and validation, user training, etc.
- Requirements are clarified by the model

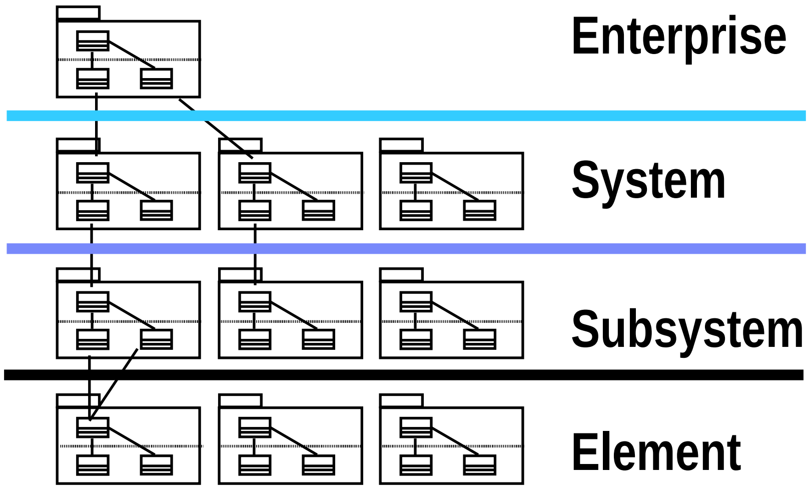


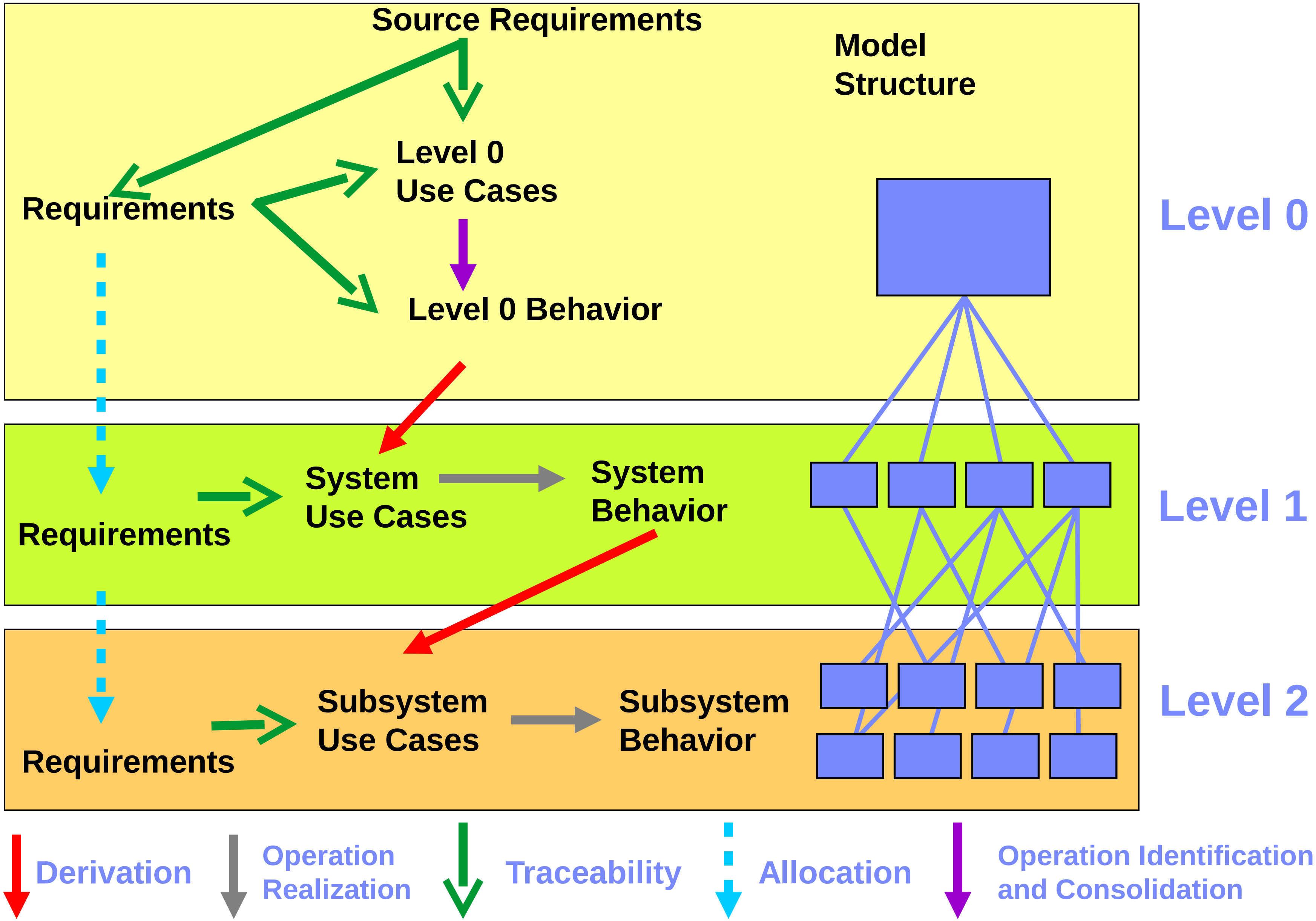
The changing role of requirements

- **Text** requirements give rise to models which elaborate and elucidate
- **Models** give rise to additional text requirements which specify and constrain
- Text and models are useful at all **levels** of system abstraction
- Capture rationale and thinking at each level to differentiate requirements from design choices

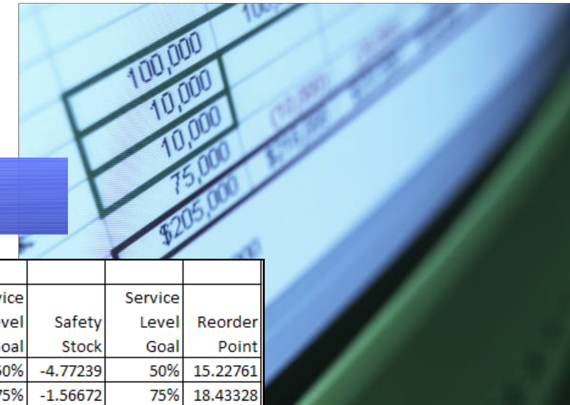
Creating comprehensive system models allows system engineering teams to reason about a more complete set of concerns, driving risk down earlier.

- The effort associated with communications grows non-linearly with the size of the project
- Capture structure and behavior; functional, logical and physical
- Show how system elements fit together.
- Keep design and implementation consistent.
- Hide or expose details as appropriate.
- Help manage complexity
- Promote unambiguous communication.
- Understand how systems are intended to be used
- Basing architecture and design on usage should produce better systems
- A system architecture provides the context for reasoning




Source Requirements
Model Structure
Level 0 Use Cases
Requirements
Level 0 Behavior
Level 0
System Use Cases
System Behavior
Requirements
Level 1
Subsystem Use Cases
Subsystem Behavior
Requirements
Level 2
Derivation
Operation Realization
Traceability
Allocation
Operation Identification and Consolidation

What makes a model a *model*?



The Dumb Way

				2080		
				2050		2150
				3450		2250
	1200	390		5575		2540
	1370	1040				3080
	610	1965		3500		
	700	700		4425		3050
	1650	1650				
	670	2950				3400
	970	640		450		3520
	1180	345				4360
	2430					5660
	4105			2600		

The Smart(er) Way

		theta=	20	sigma=	4			
[illegible]	[illegible]hi(z)	small Phi(z)	B(r)	1-B(r) / Q	Service Level Goal	Safety Stock	Service Level Goal	Reorder Point
-1.[illegible]	[illegible]642	0.19580	4.99999	0.50000	50%	-4.77239	50%	15.22761
-0.391[illegible]	[illegible]765	0.36948	2.50000	0.75000	75%	-1.56672	75%	18.43328
-0.18805	[illegible]	0.39195	2.00001	0.80000	80%	-0.75221	80%	19.24779
0.04884	[illegible]	0.39847	1.50000	0.85000	85%	0.195351	85%	20.19535
0.34486	[illegible]	[illegible]37591	1.00001	0.90000	90%	1.379448	90%	21.37945
0.77772	[illegible]	[illegible]83	0.50000	0.95000	95%	3.110889	95%	23.11089
1.25558	[illegible]	[illegible]	0.20000	0.98000	98%	5.022327	98%	25.02233
1.56891	[illegible]	0.[illegible]	[illegible]0000	0.99000	99%	6.275653	99%	26.27565

=-A11*4*(1-B11)+4*C11

=NORMDIST(A11, 0,1,FALSE)

=100+3*50+2*3*70+3*3*40+4*3*100 +5*3*125

Smarter or Dumber?
Requirements
Trade Studies
Flowdown / Allocation
Designs
Component Specifications
TPMs

- Representation of some aspect of the real world
- The relationships are baked-in
- The underlying data is flexible
- It is optimized for change

The Roles of Models 1: What are models for?

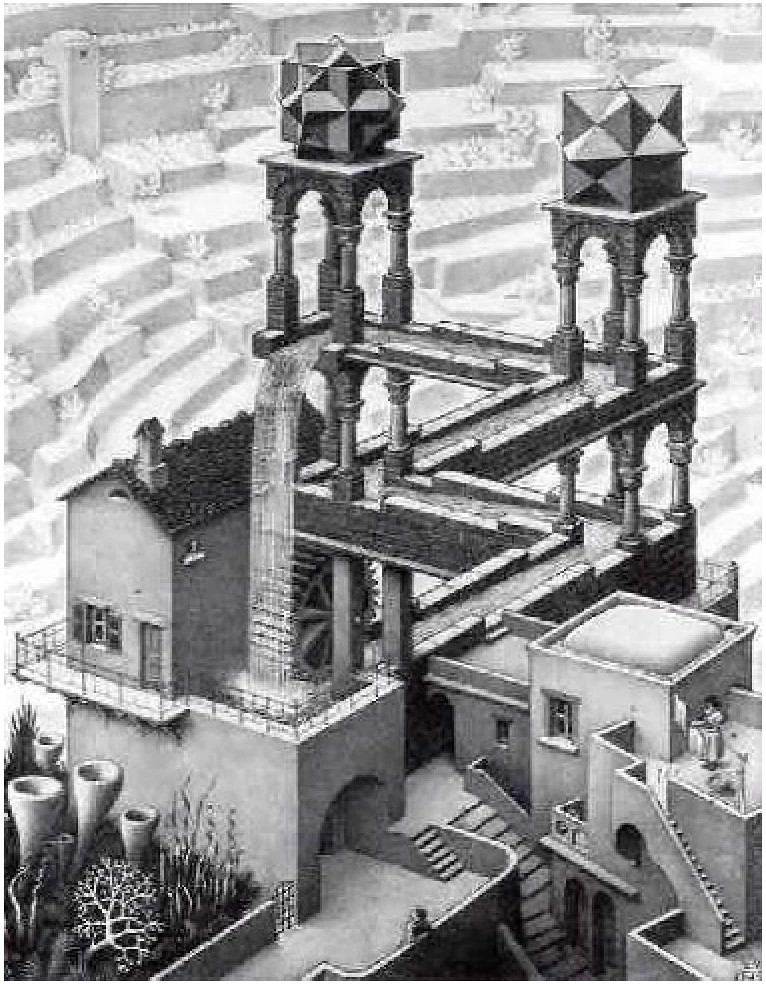
- Models can
  - **Document.** Derived from already existing reality or agreement.
  - **Represent.** Help people communicate and agree.
  - **Hold thought.** Be a shared mental space for collaborative thinking.
  - **Illustrate.** A picture is worth a thousand words.
  - **Calculate.** Both show and quantify relationships.
  - **Evaluate.** Show alternatives and criteria for trade-off analysis.
  - **Build.** Translate models into real things.
- **It is important to be clear on the purpose for a model you are creating (and maintaining.)**
  - Why are you modelling ?

The Roles of Models 2: Capturing relationships

- Models and model elements have relationships with other model elements and with other information relevant to the system.
- Important relationships
  - **Derivation**
  - **Fulfillment (coverage)**
  - **Decomposition**
  - **Dependency**
- For instance,
  - A model element may fulfill (fully or partially) a requirement.
  - A requirement may be derived from another requirement.
  - A requirement may arise from a model element.
  - A model element may decompose into another model element.
  - A model element, or requirement, may depend on another model element.
- **It is important to figure out how your model elements and requirements relate to each other.**

The Roles of Models 3: Dangers in Adopting Modeling

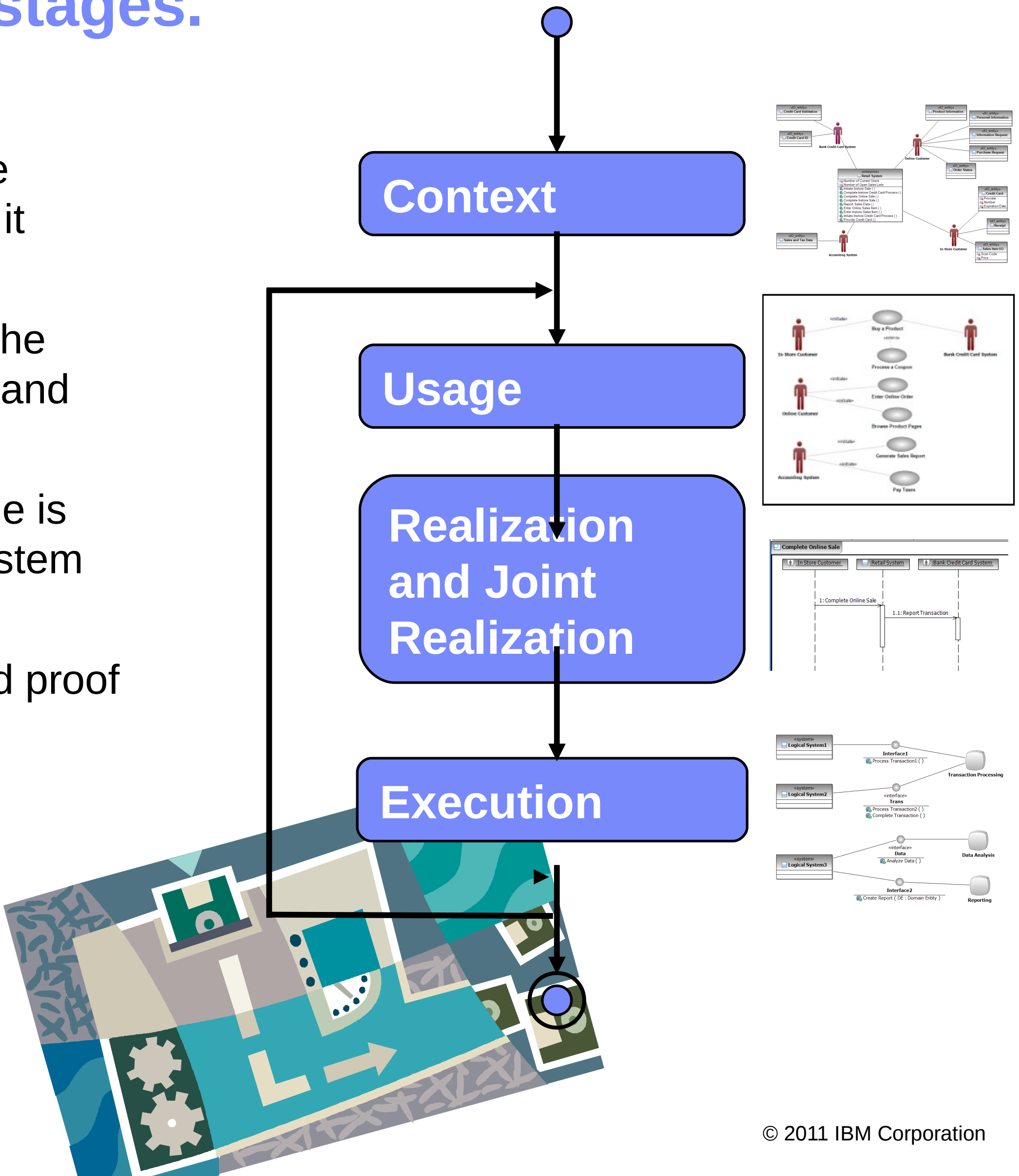
- **All models are good.** Causes model proliferation without meaning and purpose.
- **Method-centric.** We do these models because the method says so (very prevalent in DoDAF)
- **Tool-centric.** We do these models because the tool is good at it.
- **Dead models.** Non-executable models risk being un-implementable.
- **Doing documentation models exclusively.** Limits benefits of modeling (where are the real models?)



M.C. Escher, 1961

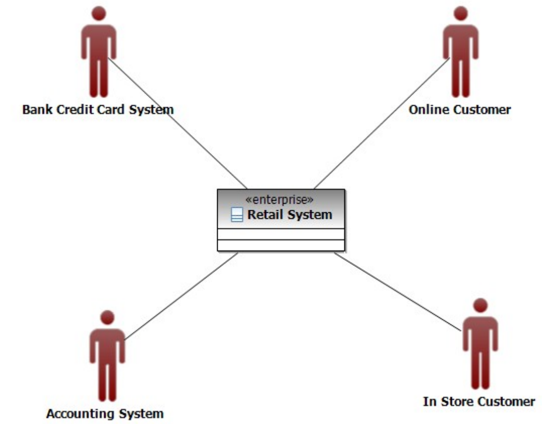
Model-driven system development models a system-of-systems in four recursive stages.

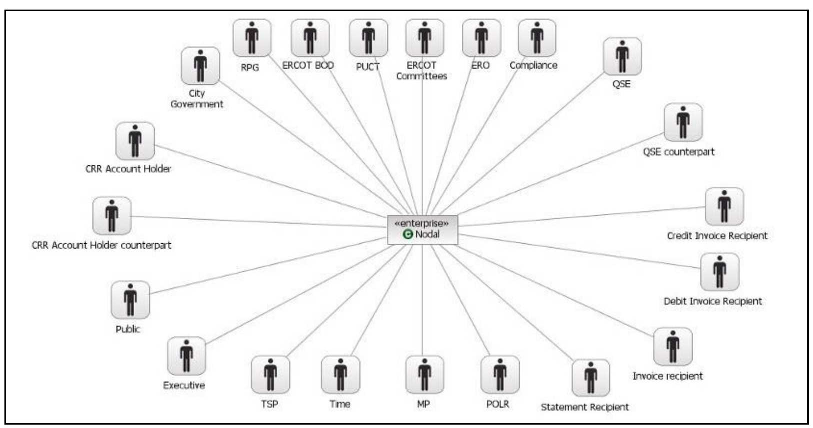
- **Context** describes the system and the people and systems who interact with it (actors).
- **Usage** describes how the actors use the system is used to produce the results and purposes of the system.
- **Realization** describes how each usage is accomplished by a collaboration of system elements using various viewpoints.
- **Execution** enables demonstration and proof of the model through execution.

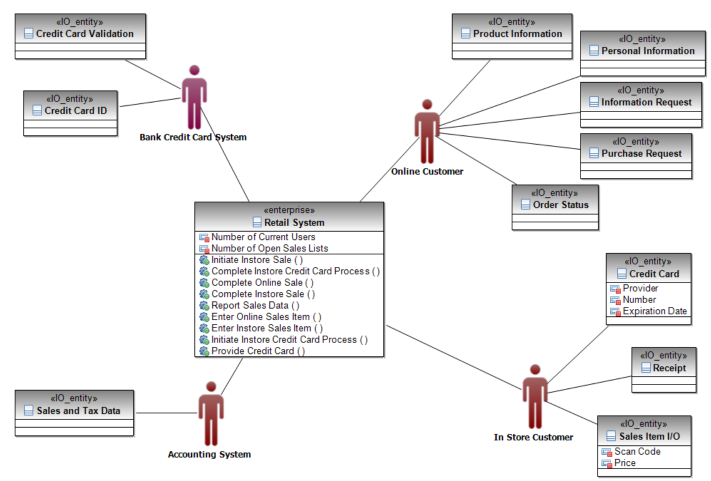


Modeling **context** allows, even forces reasoning about system boundaries high-level interactions.

- To understand systems, understand the context of the broader Enterprise
- Context Diagram treats Enterprise as a black box
- Shows scope, boundaries, operations, i/o entities
- Ultimately add operations when they are discovered through flowdown
- Add actors and i/o entities
- Shows only one level of abstraction (e.g. Enterprise)

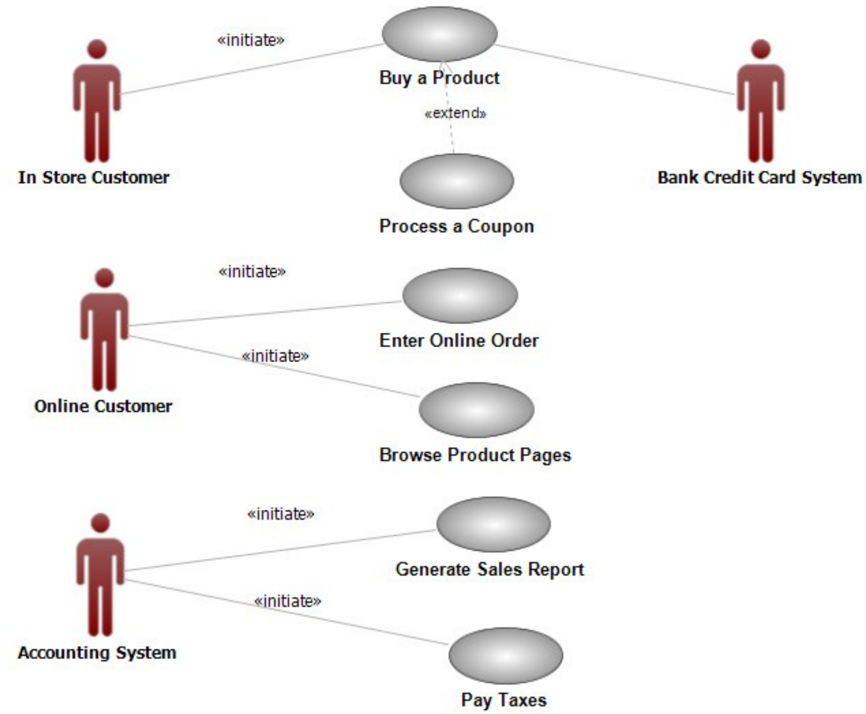






Modeling **usage** as the foundation for architecture ensures that all design serves the system's intended purposes.

- Use case: a sequence of events that yields a meaningful result of value.
- Enterprise Use Case diagram
  - "System" is the Enterprise
  - Actors same as on context diagram
  - Identify use cases:
    What do they use the system for?
- Detailed in a *Use Case Specification*
- Functional behaviour captured as an Activity Diagram or Statemachine
- Remember to treat Enterprise as a black box; no implementation details
- Focus on actor's interactions with the Enterprise
- No actor-to-actor interactions at this level: these are shown in the collaborations from the level above

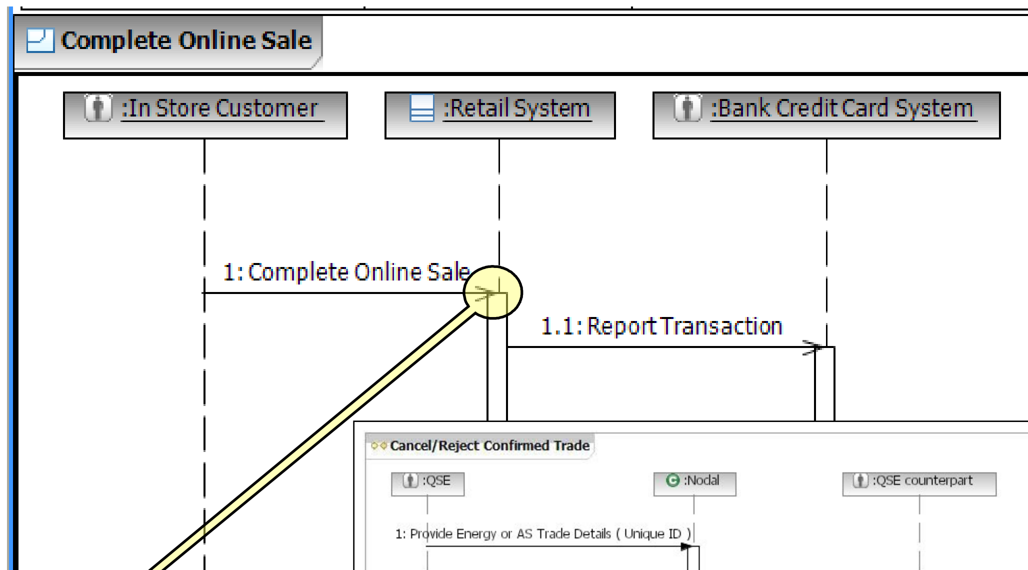


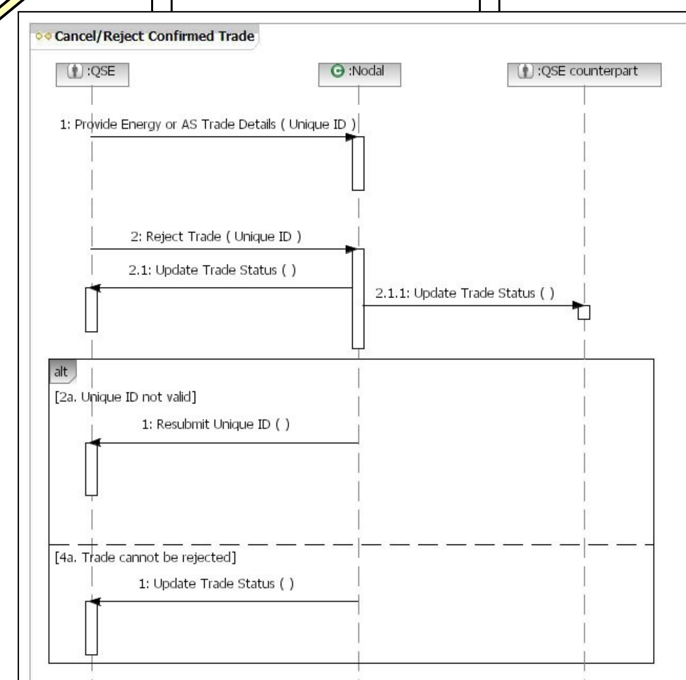
Example: GPS-Guided Travel

- GPS **Features**
  - Set destination (address, point of interest, etc.)
  - Navigate to destination
  - Retrace route
  - Get back on track
- GPS **Usage** Model
  - Find me a gas station on my way
  - What's the nearest McDonald's?
  - Is there a Hilton Hotel I can reach in about 5 hours?
  - Let's avoid freeways

Modeling **realization** connects behavior to architectural structure, integrating process with architecture.

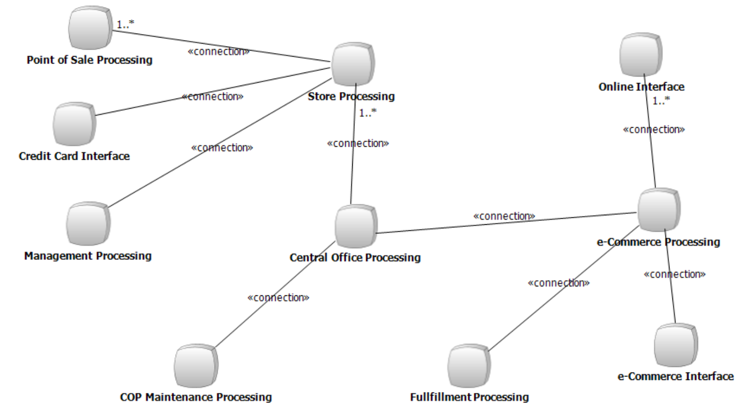
- Model enterprise as a set of operations that realize use case flows
- Highest Level “specification” of the enterprise
- Treat enterprise as black box
- Provides basis for flow down
- Identifies common logical capabilities needed across use cases
- “Black box sequence diagram” of an Enterprise Use Case allows identification of candidate Enterprise Operations.
- Name the messages as requests of the message destination NOT as the action taken by the initiator

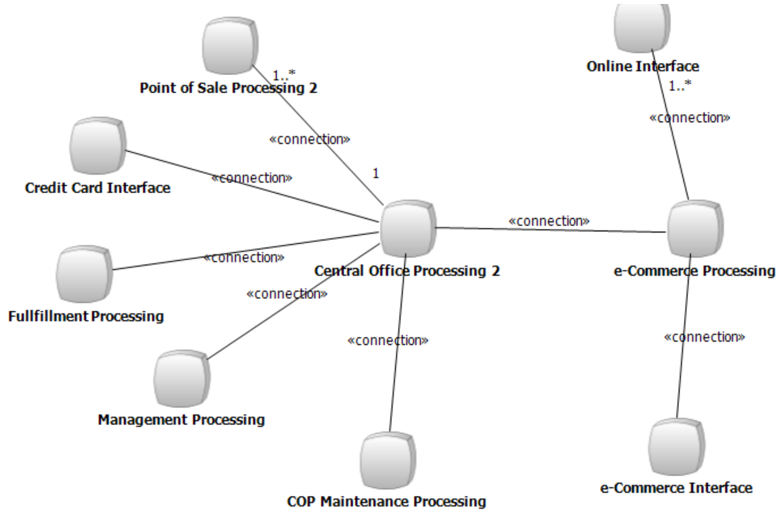




Modeling **joint realization** allows us to show the relationship between physical/distribution architecture and logical architecture.

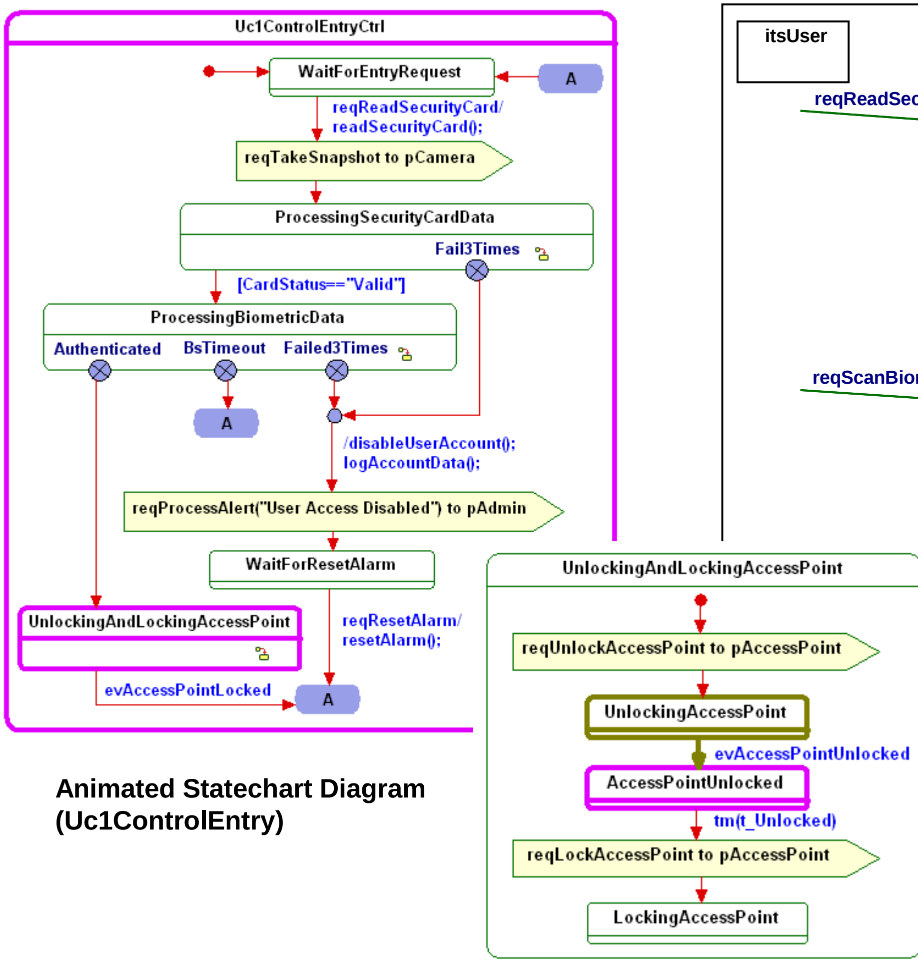
- Allocation of functionality to a location or structural element in the system
- Allows reasoning about physical distribution of system elements and functionality
  - Important for trade studies as it allows evaluation of different architectures
- A locality is a place where some system functionality happens
  - E.g. software hosted on a particular server, or an alert indication on a dashboard
- Locality diagram shows localities and their physical interconnections and characteristics.
- Localities may apply at each level of abstraction
- Locality is the context for dealing with adequacy of system to meet non-functional (quality) requirements (reliability, performance, capacity, cost, etc.)



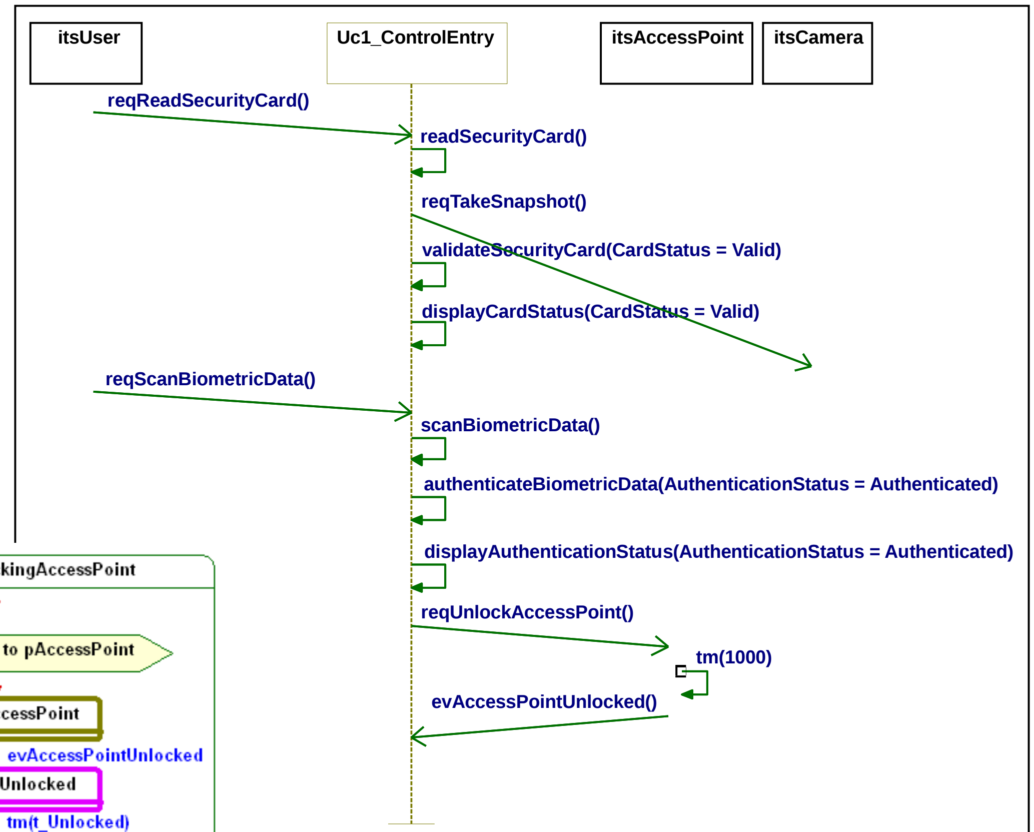


Model Execution

for early verification and validation through model execution



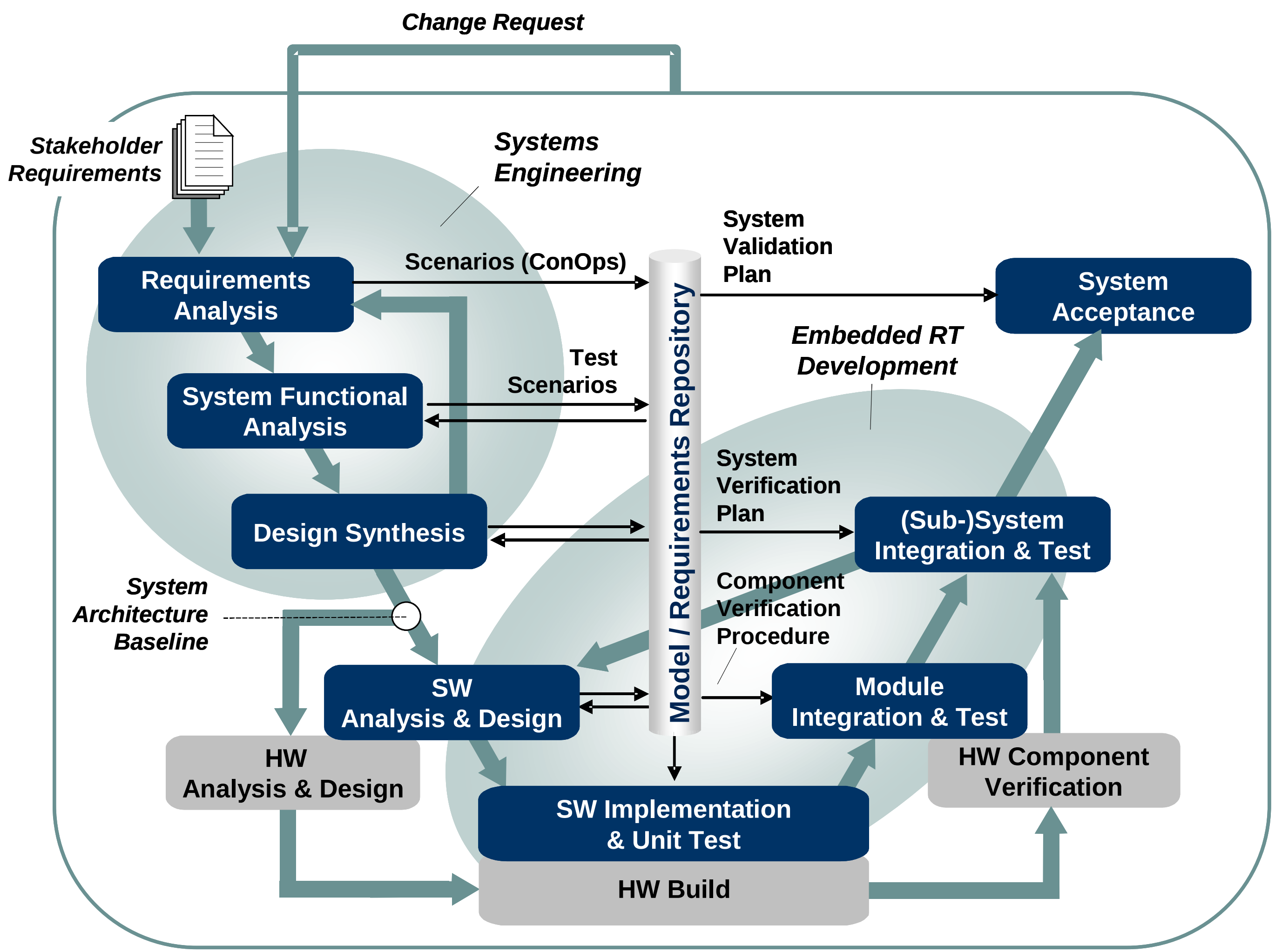
Animated Statechart Diagram (Uc1ControlEntry)



Animated Sequence Diagram (Uc1Sc1)

Integrated system / embedded software development

Design iterations in the "V" development lifecycle



Asset-Based Systems Engineering

- Reuse: doing more with less
- Moving from documentation to re-creation
- Capture engineering and business value
- Models are engineering assets
  - What, How and Why
  - Tailor and reuse
  - Contribute back to library
- Precursor to full product line approach

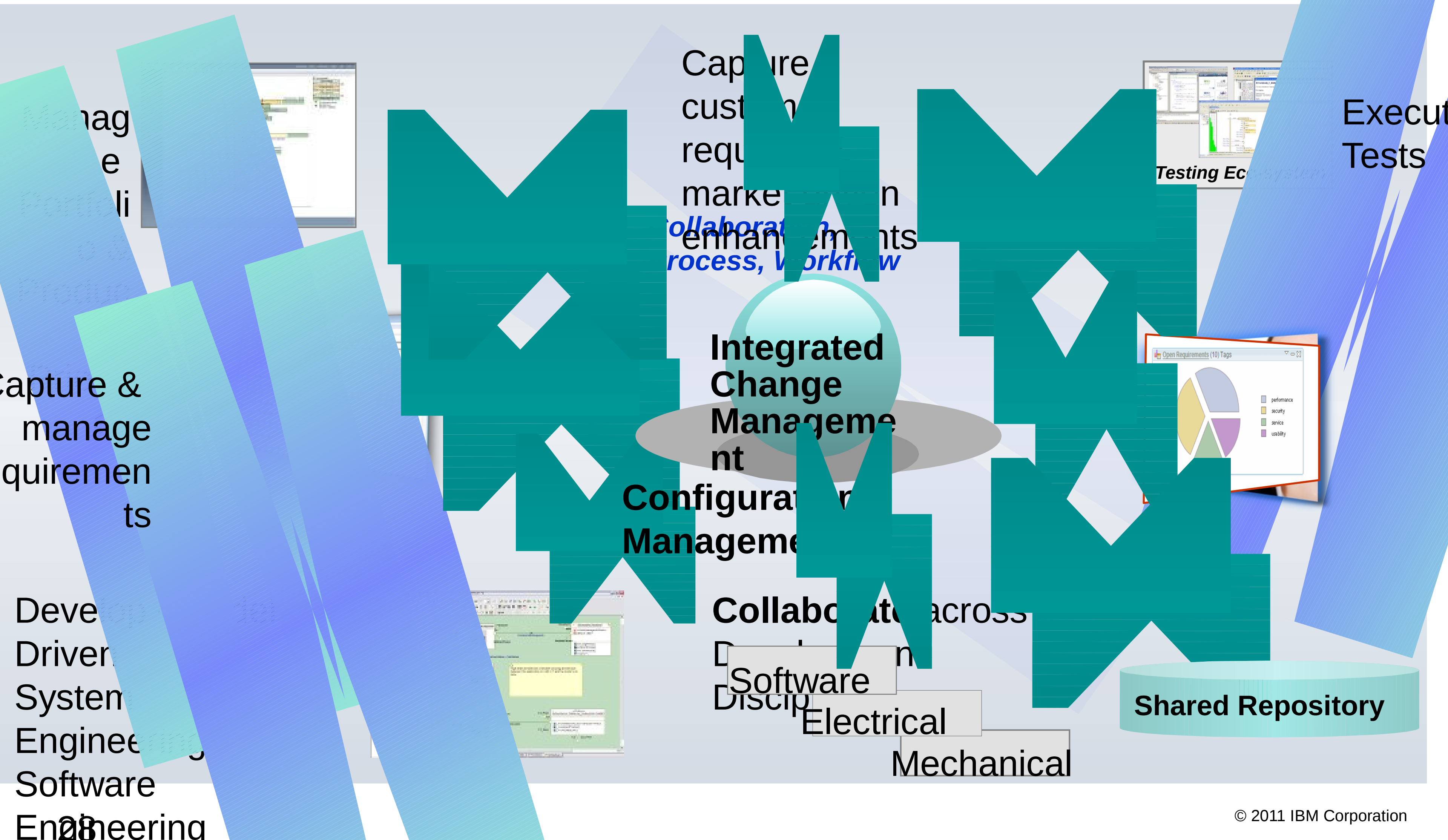
Technical Work Management:
Collaboration and Communication

Why are small teams so effective?

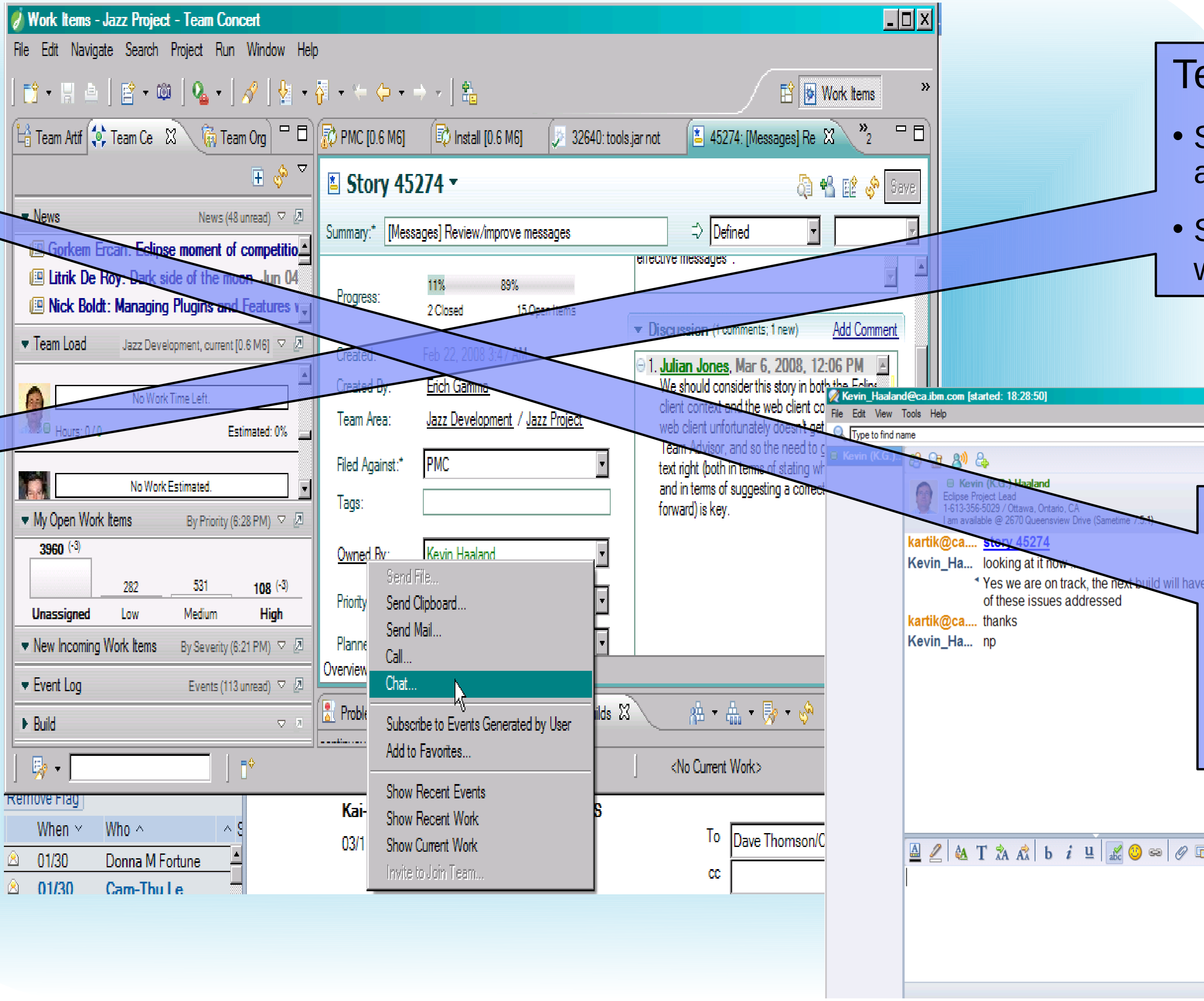
- Communication among teams facilitated by Systems Engineering
- Reduce wasted time / error associated with misunderstandings and incomplete mental pictures of system requirements
- Build common, shared work products among cross-functional teams
  - Model-based
  - Shared repositories (requirements, change requests, issues, defects, configurations)
- Coordinated, automated workflow tied to shared work products
  - Avoid email exchange of work products, multiple versions, confusion
- Build on integrated tooling infrastructure
  - Limitations of point-to-point integration
  - Jazz Platform and OSLC Vision

Technical Work Management:
Enabling Effective Collaboration

(Why is Social Media better than email?)



Collaboration in Action

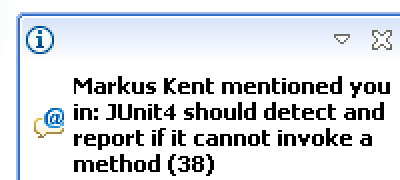


Team Awareness

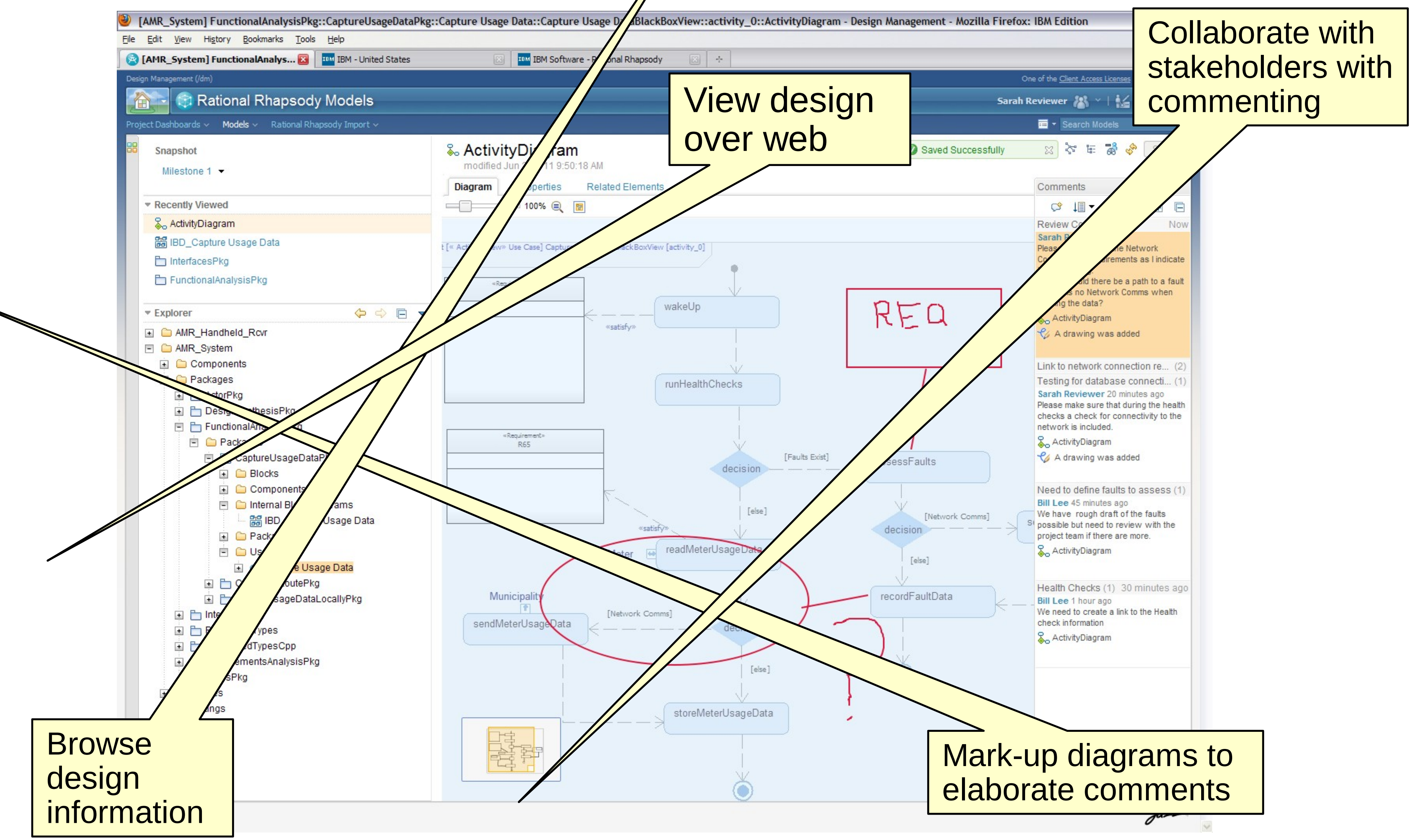
- Shows team members and their online status
- Shows what they are working on

Change Awareness

- Automatically links to changes if mentioned in chat
- Drag and drop any work item or query into chat



Enabling Collaboration with Models

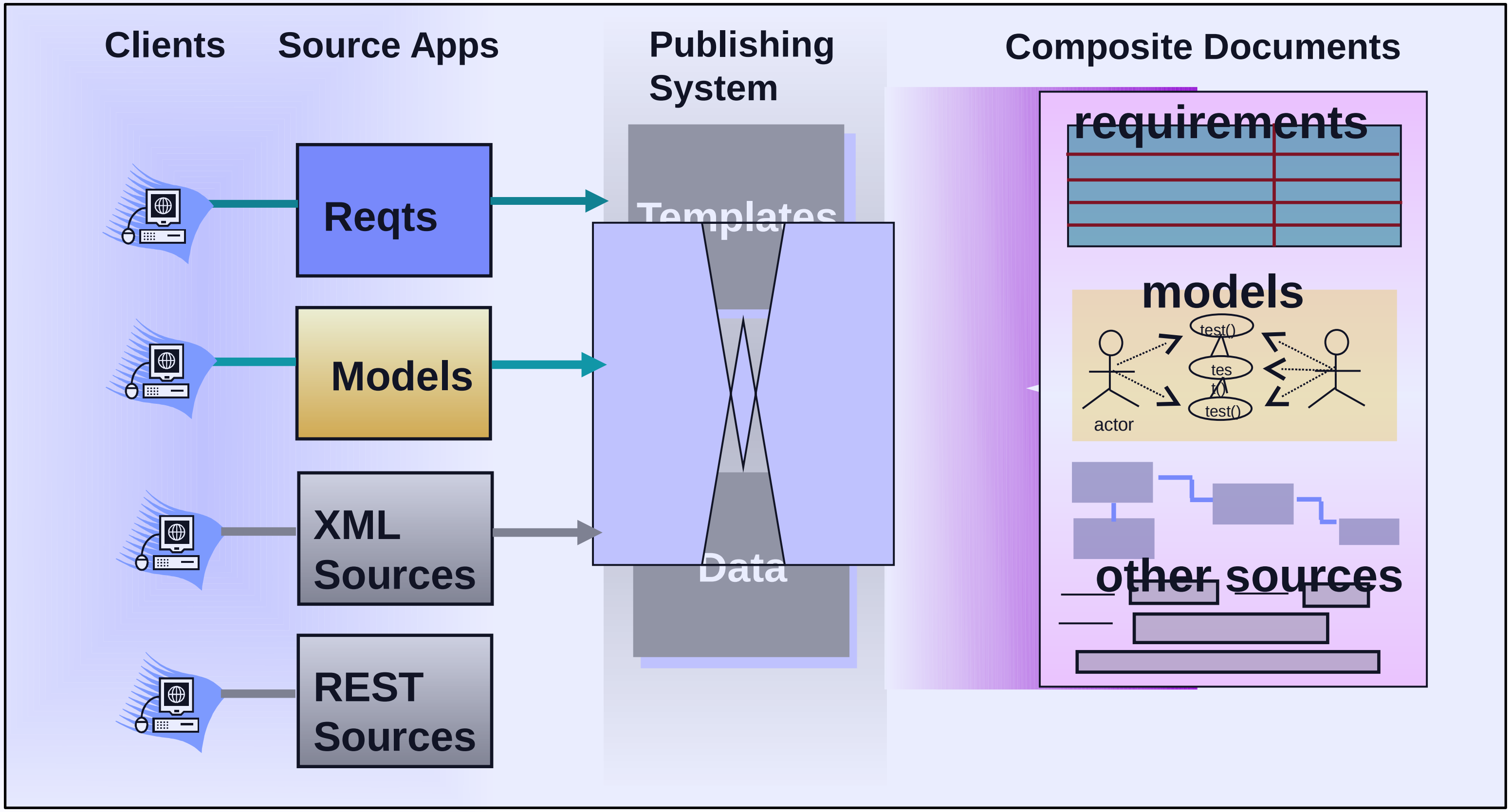


Systems Engineering traditionally floats on a sea of documentation

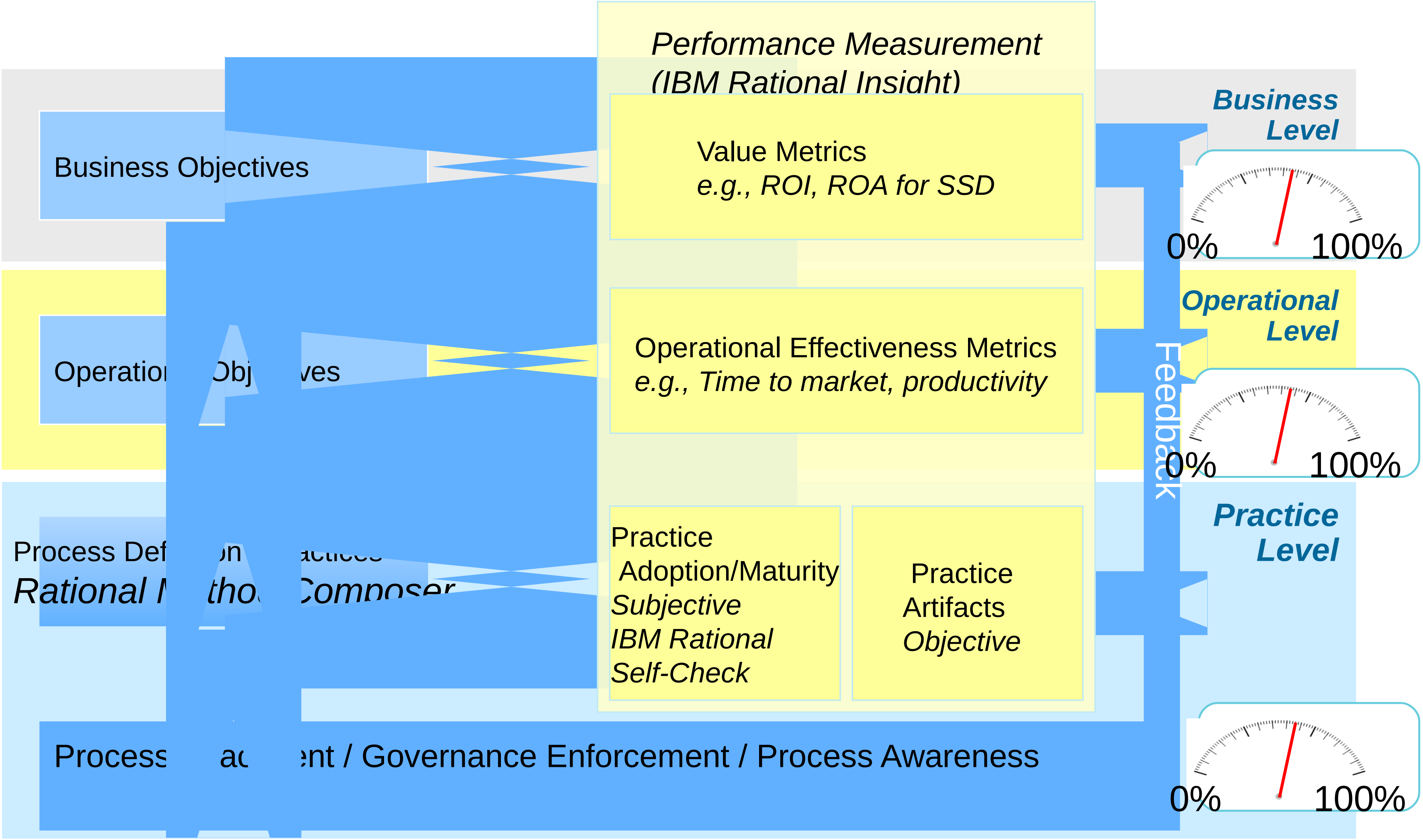
- Document-centric approach results in information dead-ends
- Changes make documents obsolete before they are approved
- Documents remain necessary for contractual management and agreement

Automated Document Production

- Treat documents as reports of live information
- Facilitate re-use and consistency

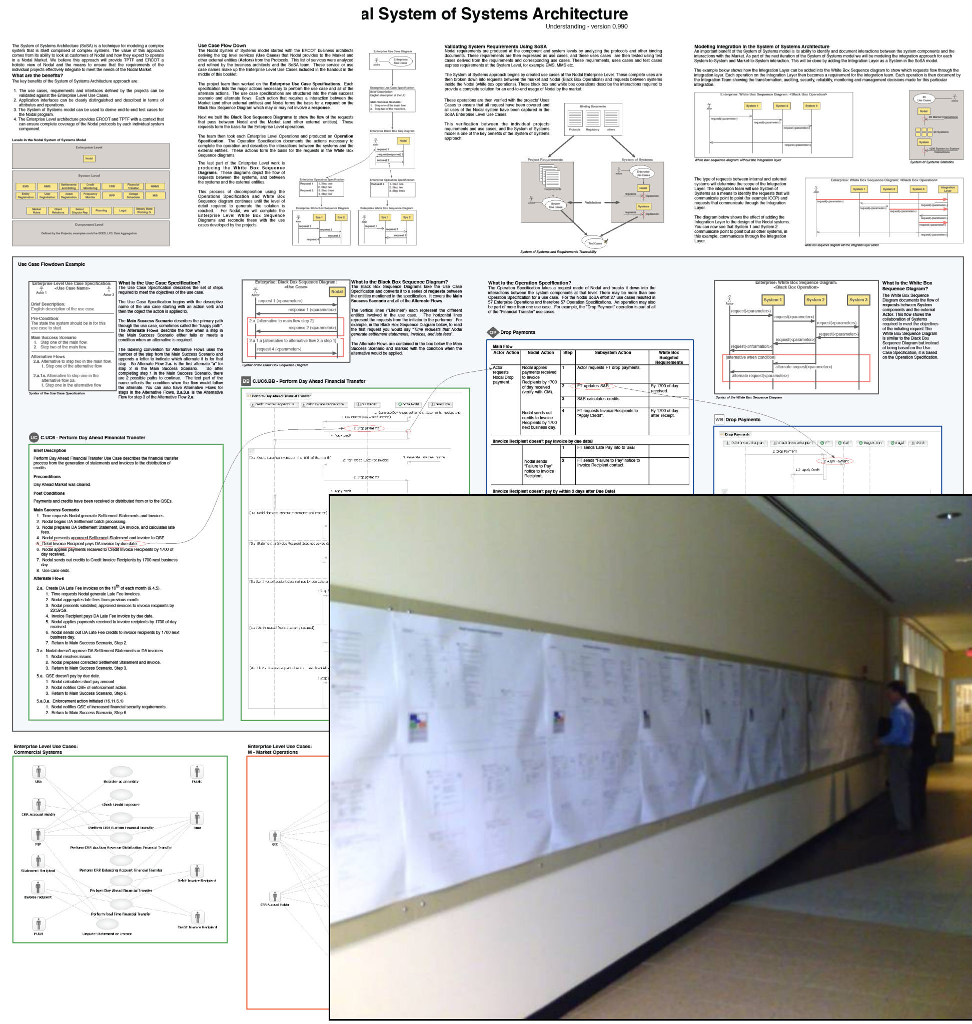


Closed Loop Governance: *Measure and Improve*



One innovative organization made their models visible to everyone in the company!

- Models first developed on flip charts
- Later maintained in automated modeling tools
- Finally, produced on large wall charts for *increased visibility*, *communication, collaboration*
- Became center for *discussion* across all teams.



- Good Systems Engineering is all about **Communication** and **Collaboration**
- Models facilitate that

www.ibm.com/software/rational

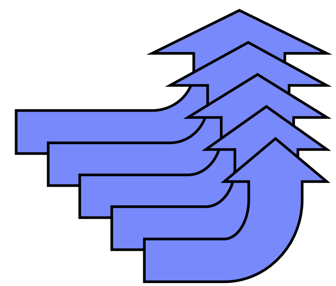
www.ibm.com/software/rational

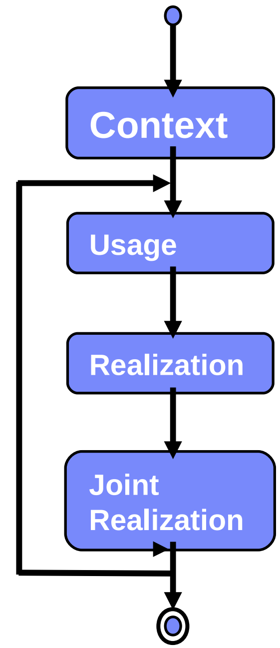
Multiple viewpoints and levels of abstraction allow system engineers to focus on specific concerns while keeping the whole system in mind.

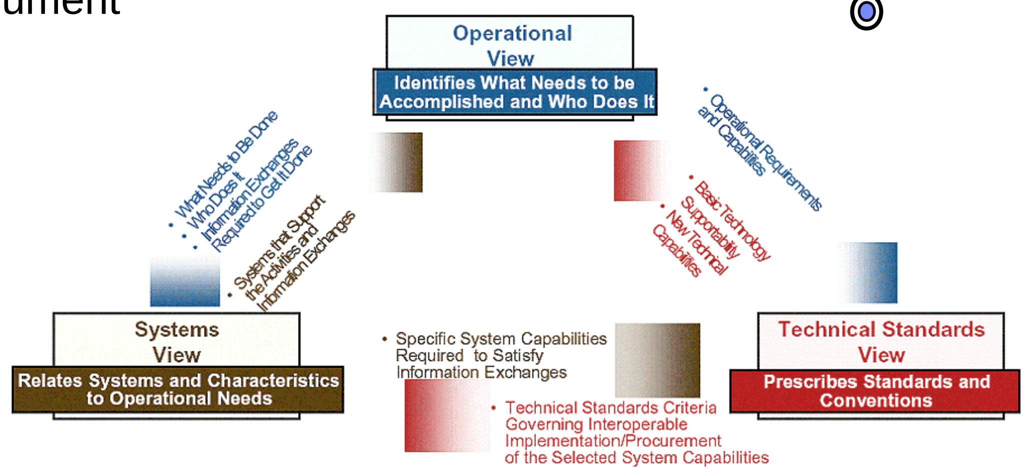
Model Level	Viewpoints				
	Worker	Logical	Information	Distribution	Process
Context	Organization View	System Context Diagram	Enterprise Data View	Enterprise Locality View	
Analysis	Generalized System Worker View	System Analysis Model (Subsystem Diagram)	System Data View	System Deployment Model (Locality View)	System Process View
Design	System Worker View	• System Design Model (Subsystem/Diagram) • Component views	System Data Schema	System Deployment Model (Descriptor View)	Detailed Process View
Implementation	Worker Role Specifications and Instructions	Deployment diagram at Implementation level for each configuration			

Standard architectural views like DoDAF/MODAF aid in communication to stakeholders, and should derive from the architectural models.

- High-level stakeholders can learn to expect certain high level views
- Identify, classify and harvest DoDAF content in the form of reusable assets
- Automated matrix creation to ensure consistency
  - AV-2, OV-3, SV-3, SV-4, SV-5, SV-6
- Reporting of all DoDAF views
  - Generated to a Microsoft® Word document
- Import custom graphics (OV-1)
- DoDAF model framework for work products and views







Value of Systems Engineering: Success and Quality

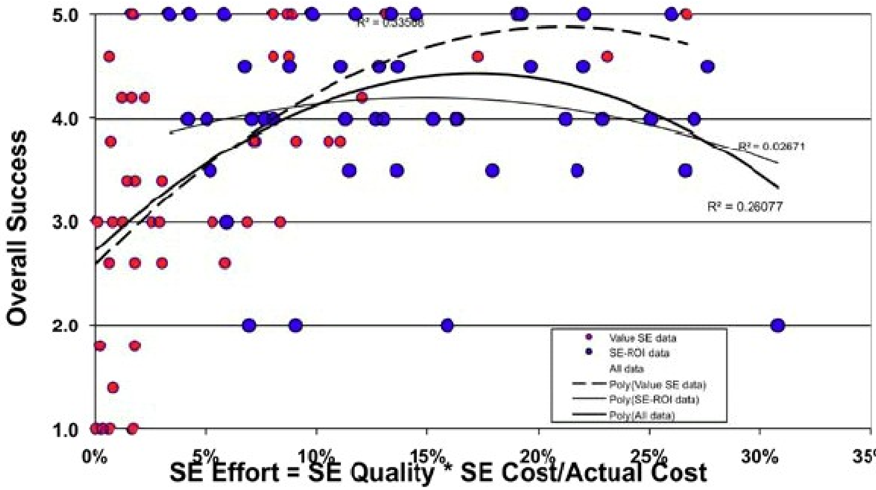


Figure 12. Correlation of SE Effort to Subjective Succe

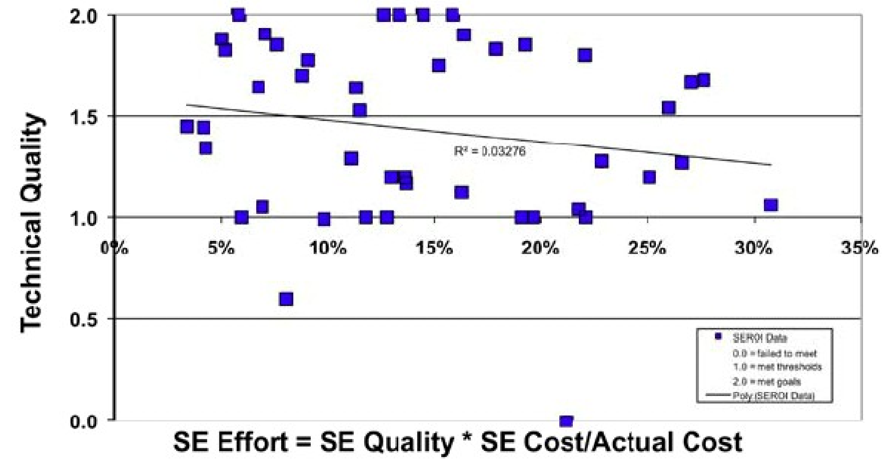


Figure 13. Correlation of SE Effort to Technical Quality

Applying the right amount of systems engineering is critical to program success.

Source: Honour, Eric (2010), *Systems Engineering Return on Investment,* University of South Australia, p9

Joint realization shows how operations are realized both in logical and physical architectures.

- Select level of abstraction for locality model
  - Not usually done at Level 0 or 1
  - Maybe done at multiple levels
- Example based on doing sequence diagram of a use case
  - As before with logical elements
  - Now with physical elements
- Shows physical elements and actors
- Show each operation as message into physical element.
- This can also be done with activity diagrams and swimlanes