NDIA Paper:
Use of a Model-Based Approach to Minimize System Development Risk and Time-to-Field for New Systems

Wagner, Brockwell, Daniels, Loesh, Gosnell

09 October, 2010

Overview

- Who we are – AMRDEC Software Engineering Directorate (SED)
- Overview of current state of project development with statement of the problem
- Statement of Problem: Some Significant Root Causes
- Solutions to Avoid/Solve the Problem to Achieve Project Success in Minimal Cycle Time
- The process – organization, structure, workflow
- Demonstration & metrics from two SED projects
- Summary & Conclusion

Abstract

- The DOD industry is constantly seeking ways to decrease the time-to-field for new technology and improved systems to meet war fighter needs. But instead of decreased cycle-times, often development times are increased due to new requirements for system information assurance, safety, interoperability, and other technology and certification issues. Increasingly, standards for information assurance and safety certification require the early involvement of specialized safety and IA teams in the development process. An increased number of evaluation and test artifacts must also be produced by product development teams. These additional, required artifacts add substantially to system development time and costs.
- As a result, the Army AMRDEC Software Engineering Directorate (SED) is increasingly asked to provide help in developing systems with full engineering rigor, while achieving lower system life-cycle costs and shorter time-to-field schedules that cannot be achieved using other formal acquisition strategies. SED has met the need using an integrated, model-based development approach. We are applying state-of-the-industry modeling and requirements management/development tools and technologies to shorten development times, improve system and software reliability, and satisfy increased requirements for system safety (e.g. DO-178B), security (e.g. EAL-6), and interoperability. Code generation algorithms provided by modern UML-based modeling tools can be tailored to meet the coding guidelines imposed by standards for software safety. Additionally, requirements and design documents can be generated more reliably, and with substantially reduced cost and schedule impact. Early requirements and system architecture verification is achieved through model execution, thus correcting errors early in the development cycle and avoiding associated schedule impacts. The net effect is a shorter time-to-field development cycle, while retaining a high degree of engineering rigor and compliance with SED's mature processes.
- This paper discusses the approach and the results achieved using it to develop a high-risk, short-lead time, fielded system.

Overview

- Who we are – AMRDEC Software Engineering Directorate (SED)
- Overview of current state of project development with statement of the problem
- Statement of Problem: Some Significant Root Causes
- Solutions to Avoid/Solve the Problem to Achieve Project Success in Minimal Cycle Time
- The process – organization, structure, workflow
- Application & Metrics from two SED projects
- Summary & Conclusion

AMRDEC SED Introduction: Who we are and what we do

- We are the U.S. Army Aviation and Missile Research, Development, and Engineering Center (AMRDEC) Software Engineering Directorate (SED).
- We work to support a very diverse set of engineering life cycle areas:
  - Technology Development (Pre-Milestone A & Pre-Milestone B)
  - System life-cycle management (Post-Milestone C)
  - Formal System Development (Milestone B to Milestone C)
  - System Verification, Validation, Certification, Qualification (Milestone C transition)
- Project Role Varies
  - Function as Materiel Developer of Fielded and Support Capabilities
  - Function in Support of Acquisition Agent (PM) to Assure Project Success
  - Act as Supporting Independent Qualification/Certification/Integration Agent
  - Operate In-house Facilities in Support of System Operations and Use
- Projects Span Full Range of Size
  - Projects range from 2-3 engineers to 2-3 hundred engineers
  - Project development products from few thousand SLOC to almost 10 million SLOC

AMRDEC SED Introduction: Nature of Our Programs

- Projects Span Full Spectrum of Army Strategic & Tactical Capabilities
  - Communications (VMF Parser, JTRS, tactical communications)
  - Common Infrastructure (ASE, Common Radio Control, SOSCOE, others)
  - System Exploration and Architecture Support
  - Assets for Integration Qualification, Certification, Interoperability Testing (Interop Lab)
  - Aircraft SILs, Aviation Support, and Avionics Development (ASIF, SILs, ANMP IEC,MFOQA)
  - Ground Launchers (MLRS, NLOS, THAAD)
  - Missiles
  - Strategic and Tactical Radar (SBX, other)
  - Specialty Avionics (Survivability, Situation Awareness, Navigation & Control, CBM)
  - Command & Control (JBC-P, others)
  - Specialty Projects

Extremely Diverse & Full Spectrum SE Capabilities for Equally Diverse Set of Projects Represents Significant System Engineering Challenges

Overview

- Who we are – AMRDEC Software Engineering Directorate (SED)
- Overview of current state of project development with statement of the problem
- Statement of Problem: Some Significant Root Causes
- Solutions to Avoid/Solve the Problem to Achieve Project Success in Minimal Cycle Time
- The process – organization, structure, workflow
- Application & Metrics from two SED projects
- Summary & Conclusion

Statement of the problem

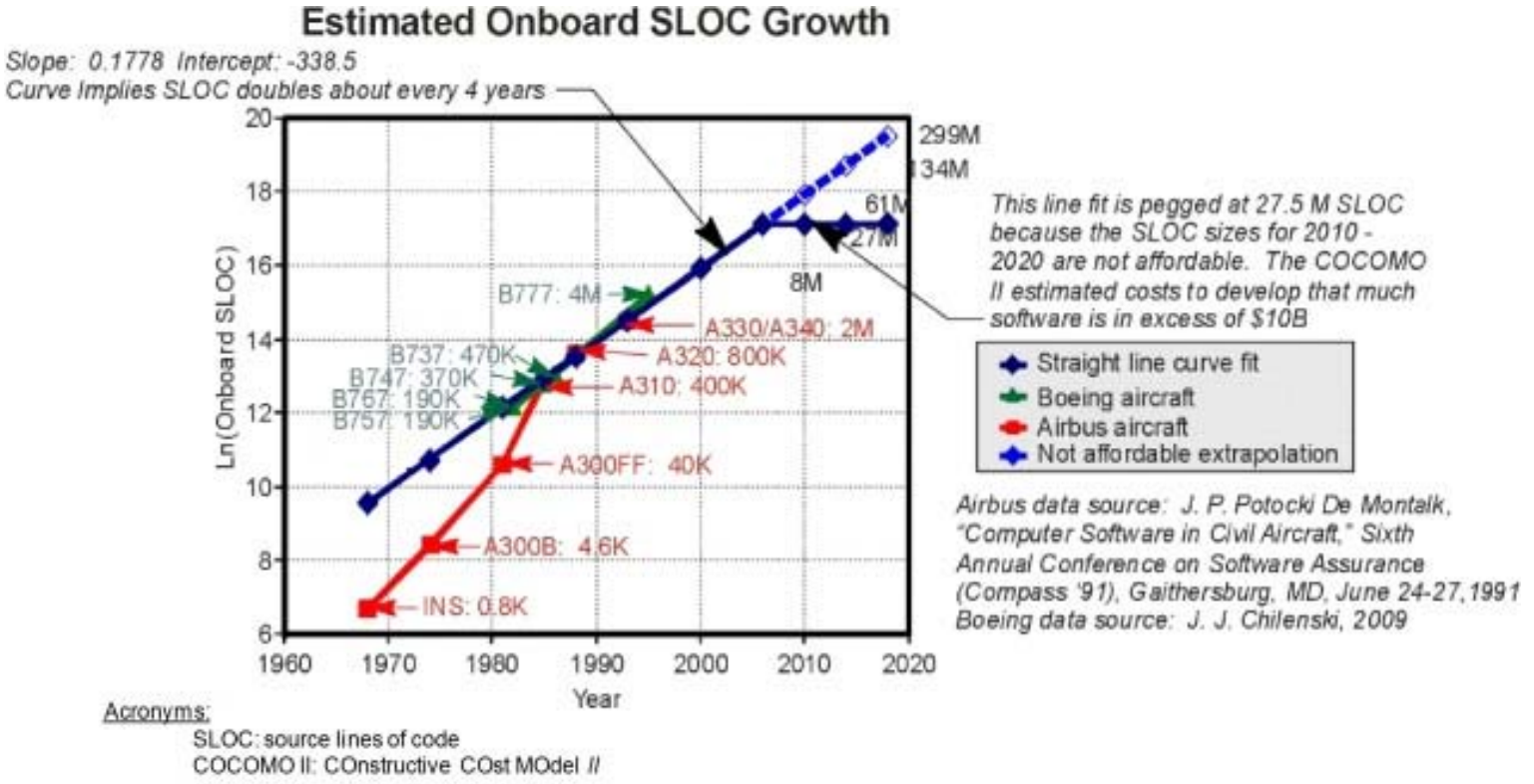
- The time to field new systems is, more often than not, unnecessarily long
- Designs often don't meet the needs when fielded
- To compound the problem typical embedded systems are increasing in complexity at exponential rate
- And all too often even with long program schedules, programs don't meet planned milestones.
  - This is further complicated by technical partners getting out of synchronization on large multi-development teams
- It is not often realized that there are actually two development efforts being executed for each system being developed:
  - The actual System Under Development (SUD)
  - The integration and test system
  - The second system is as (if not more) important but is often treated with substantially less rigor and focus
  - The test system must accompany the system into deployment as part of the necessary life-cycle support infrastructure

Statement of the problem (Continued)

- System integration and verification times are typically excessive and fraught with rework - which further compounds schedule achievement.
- Certification times required for fielding and deployment often cause schedules to slip. Many DOD systems being fielded now require certification for more than just qualification purposes:
  - Airworthiness
  - Validation/Qualification
  - IA/security
  - Safety
  - Interoperability
- There are other problems too numerous to mention but the bottom line is:

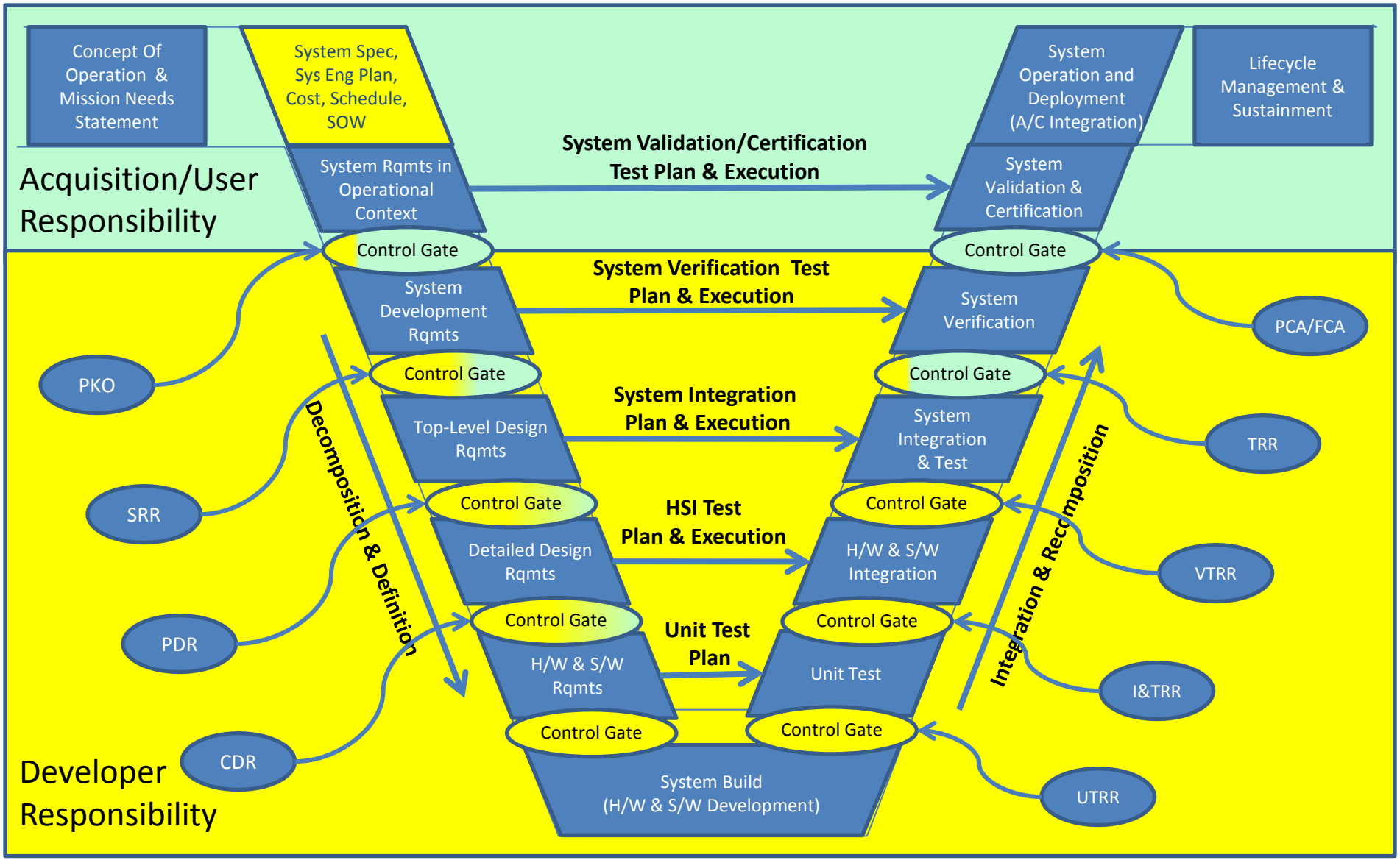
It takes too long to get systems into hands of the soldier

Exponential Growth of System Complexity*



* System Architecture Virtual Integration: An Industrial Case Study, November 2009, TECHNICAL REPORT CMU/SEI-2009-TR-017 ESC-TR-2009-017

The Development "V": Errors Introduced in Decomposition Cause Rework during Composition

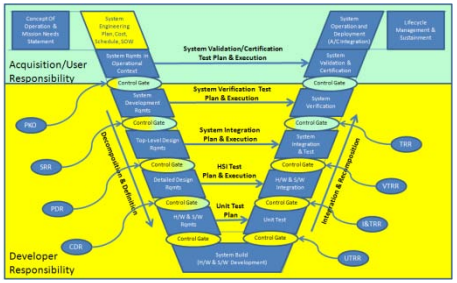


Overview

- Who we are – AMRDEC Software Engineering Directorate (SED)
- Overview of current state of project development with statement of the problem
- Statement of Problem: Some Significant Root Causes
- Solutions to Avoid/Solve the Problem to Achieve Project Success in Minimal Cycle Time
- The process – organization, structure, workflow
- Application & Metrics from two SED projects
- Summary & Conclusion

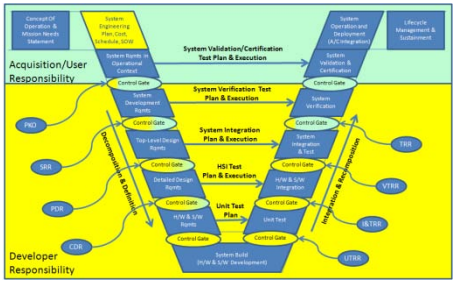
Root Cause 1:
Error Propagation During Decomposition

- System development is typically characterized notionally by the “V” even where non-waterfall methods are used.
  - It is a set of “black box” to “white box” design synthesis iterations, each iteration comprising a “decomposition”. Iterative decompositions are performed until the system design is specified to a level appropriate to begin building the lowest level system elements. This is called “going down the decomposition side of the V”
  - A set of “composition”, integration, and verification activities, starting at the lowest level elements, is performed to ensure that the finished product meets it’s specification at each level. This iterative “composition” takes place at successively higher levels as the system is integrated and tested. This is sometimes called “coming back up the composition side of the V”
  - Errors creep into each and every design synthesis iteration – practically impossible to avoid!
  - And these errors are typically not detected until the composition activities of integration and verification testing, when correcting them is the MOST EXPENSIVE



Root Cause 2:
Immature Test Environment at Start of Composition

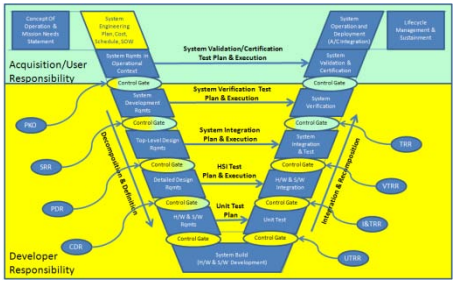
- During each iteration of the design synthesis activity, when a black-box is decomposed into white-box elements and associated requirements, the resulting requirements are traced up to those of the associated black box
  - Thus a full set of the required bi-directional requirements traceability links are generated.
- A set of requirements is generated for testing each of the resulting white box elements thus creating a set of horizontal requirements links to the test capabilities that will be needed later.
  - Elements of the project Integration & Test (I&T) work products are critical for testing a wide spectrum of attributes: safety, reliability, IA/security, performance, interoperability, other.
  - The test system (capabilities, labs, fixtures, design requirements, test specs, test procedures, etc) must be developed in parallel with the development of the SUD.
  - The I&T work products are typically not matured early by synergistically using them to evaluate the early “bottoms up” prototypes.
  - Failing to develop a mature test infrastructure and/or vetting it BEFORE coming up the composition side of the V causes EXPENSIVE and TIME CONSUMING schedule impacts during the test sequences later in the project.



Root Cause 3:

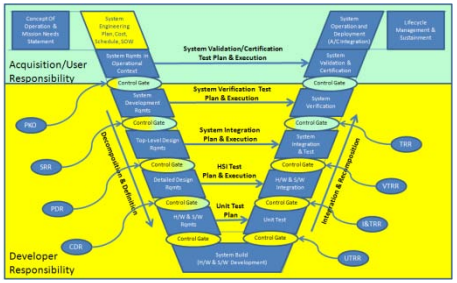
Insufficient Design Trade Analyses Cause Faulty Designs

- Due to the labor-intensive nature of the additional efforts to perform REAL risk reduction and the required design trade analysis (which often requires prototyping) the system decomposition does not proceed in a way that allows the effective achievement of the required Measures of Effectiveness (MOE s)/Measures of Performance (MOPs). Redesign often occurs later in the development phase as a result of poor early design decisions.
  - During composition, rework is caused by finding errors in system implementation activities as well as by finding errors in DESIGN. Design errors are substantially more expensive than implementation errors
- If design trade analyses are performed by prototype development, then the associated risks are mitigated. BUT:
  - Prototypes tend to be very expensive and prone to failure due to nature of developing hardware and software early in a less structured development environment
  - Plus these prototypes are usually throw-away since they don't meet requirements for safety, security, reliability, etc.
  - Thus additional work must be done associated with bringing the "proof of concept" prototypes up to objective system quality



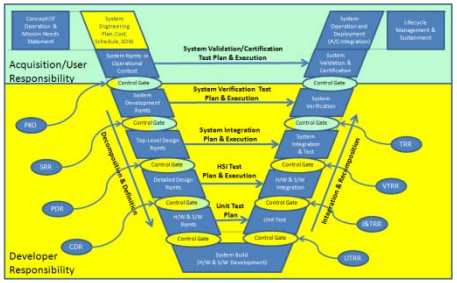
Root Cause 4:
Certification/Qualification Issues Discovered Too Late

- Software source code is usually not available early enough in the project to get early assessments of software compliance with coding standards for safety, reliability, security, and information assurance.
  - When deferred until later in the development activities when software is available, rework is required to correct compliance issues and, in many cases, this rework causes redesign and substantial impacts to development schedules.
  - Failing to develop software to objective requirements necessary for certification/qualification causes EXPENSIVE and TIME CONSUMING schedule impacts due to software redesign, retest, reintegrate (extensive rework).



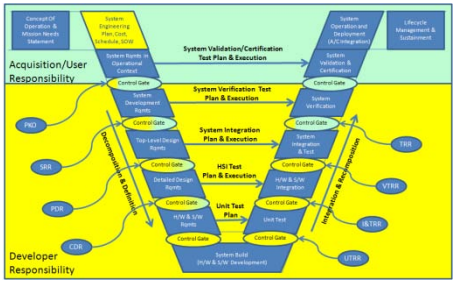
Root Cause 5:
Project Schedule Critical Paths are not Optimized Properly

- Often system development schedules are not sufficiently parallelized to take early advantage of system capabilities that are well known.
  - Projects often do not take advantage of efforts to parallelize development that result in early system capabilities. Failure to pick the low hanging fruit early does not take advantage of using the well understood system capabilities to mature understanding of more difficult or high-risk system areas.
  - Problems associated with this are that early in the project mature development and test infrastructure can sometimes inject errors and immaturity in areas of criticality (safety, security, information assurance, etc)
  - Same problems with early prototyping efforts
  - This problem is related to the lack of effective Risk Management (RM)
  - Should always go Top-Down and Bottom-Up concurrently and those efforts should be risk-mitigation-driven

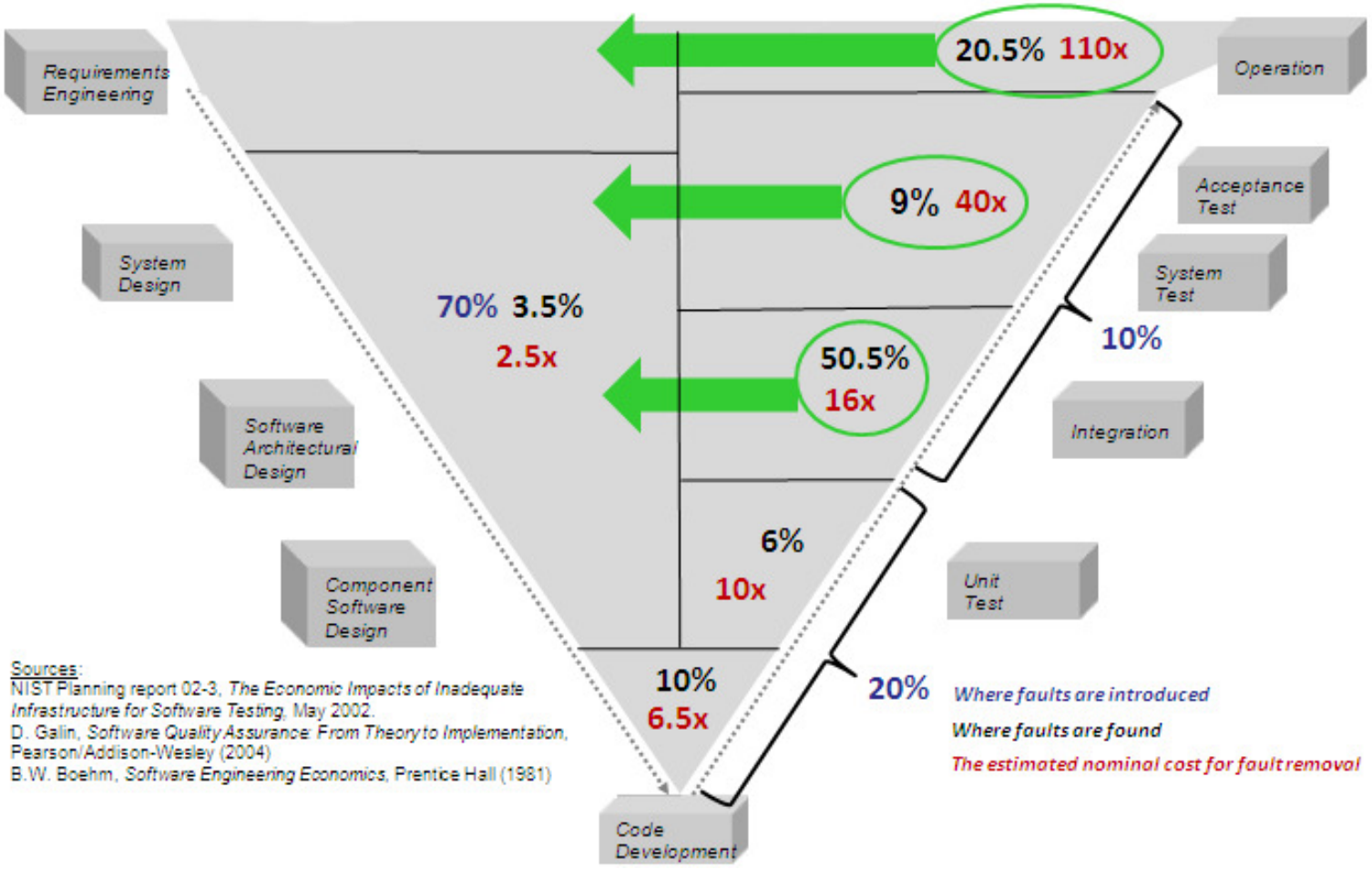


Root Cause 6:
Automated Development Tools not Often Leveraged

- Modern tools for system development (system architecture, design, performance, complex electronics) that could drastically shorten project timelines when they are integrated with the engineering processes of the enterprise are not often used.
  - Tools that provide automation are not typically used and …
  - When modeling and development tools are used they typically are used in a "stove-piped" manner and not integrated with each other or into the project's system engineering processes
  - This is the old "John Henry and his sledge hammer" syndrome: manually generating software only because it is trusted and the results (however costly and time consuming) are known.
  - Manually attacking system engineering tasks further slows things down and typically injects additional errors.
  - Cannot possibly test comprehensively with manual, discrete analyses and test efforts



Impact of Error Propagation During Decomposition Phases*



* System Architecture Virtual Integration: An Industrial Case Study, November 2009, TECHNICAL REPORT CMU/SEI-2009-TR-017 ESC-TR-2009-017

Summary:
Problems in Achieving Shortened Development Cycles

- It should be evident by now that significant cost and schedule impacts are due to:
  1. Error propagation during decomposition: Errors are not detected until the composition activities (integration and verification testing) - REWORK
  2. Untested and immature integration, verification and test capabilities, and work products (such as test specs, test case design, test procedures) – SLOWS DOWN INTEGRATION AND VERIFICATION TESTING
  3. Insufficient Design Trade Analyses which cause faulty designs resulting in expensive rework later
  4. Certification/Qualification Issues Discovered Too Late
  5. Lack of properly making project activities parallel where possible – DON'T WAIT UNTIL THE LAST MINUTE TO DEVELOP A CAPABILITY THAT IS WELL UNDERSTOOD
  6. Lack of properly recognizing and mitigating system risks early cause delays resulting from "mini-cycles within major development cycles" – LEADS TO MORE REWORK
  7. Failure to leverage automated techniques resulting in development efforts that are too often labor intensive and result in more error injection

Overview

- Who we are – AMRDEC Software Engineering Directorate (SED)
- Overview of current state of project development with statement of the problem
- Statement of Problem: Some Significant Root Causes
- Solutions to Avoid/Solve the Problem to Achieve Project Success in Minimal Cycle Time
- The process – organization, structure, workflow
- Application & Metrics from two SED projects
- Summary & Conclusion

Solutions for Achieving Shortened Development Cycles

- Problems can be addressed by:
  1. Eliminating design/decomposition error propagation by fully testing and vetting system design requirements in their operational context at EVERY level of requirements analysis and design synthesis USING A FULLY FUNCTIONAL SYSTEM AND OPERATIONAL ENVIRONMENT SIMULATION – THE SYSTEM MODEL
  2. Fully parallelizing project development activities by performing design trade analyses in parallel with the design synthesis (decomposition) activities at every level using THE SYSTEM MODEL instead of costly physical prototypes.
  3. Vetting "black box" requirements BEFORE beginning decomposition to "white box" elements by testing within THE SYSTEM MODEL, which represents the combined behavior of the "white box" elements within THE SYSTEM ENVIRONMENT MODEL
  4. Using a modeling tool to generate THE SYSTEM MODEL that contains automatic code generation capabilities, so that as the model is verified at each design iteration the software source can be given to supporting specialty teams assessing certification. Code generation rules can be modified to meet required standards for security (EAL-4, EAL-6, DO-178, etc)
  5. Developing the integration and test infrastructure and verifying its operation by concurrently developing THE SYSTEM OPERATIONAL ENVIRONMENT MODEL with THE SYSTEM MODEL of the system under development.

But…

- We have characterized some of the problems with development failures, and we are addressing them
- At the heart of our solution is a model-based architecture development approach to solve these complex problems.
- But it is not as simple as just buying a model-based development tool and training staff and letting them go.
  - The tool must be integrated into the enterprise processes and used to support the needs of the project
  - There must be a strong, rigorous, and disciplined system engineering set of processes to supplement the tool.
- Other salient aspects of the solution:
  - Use of a strong interdisciplinary team to support concurrent engineering processes and practices.
  - Making parallel as much of the development as possible to avoid long critical paths in program execution
  - Utilization of a top-down and bottom-up engineering effort that strongly leverages prototype development to support risk management (RM) and decision, analysis, and resolution (DAR)

Integrated Model-Based Development Overview

Acquisition/User Responsibility

Operational Requirements

Start here!

MOEs/MOPs

TPMs/KPPs

System Attributes

System Model

Behavioral and Performance Based

System Environment (Complete System Operational Context)

System

Test Scenarios and Cases

Test Results & Performance Measures

- Use Cases
- Sequence Diagrams
- State Transition Diagrams
- Activity Diagrams

Establish values of model parameters to support model for execution to Support Design Trades

Prototype H/W

Prototype S/W

Analysis Model

System Operational Environment (Airplatform, et.al.) — System

Architectural Candidate 1

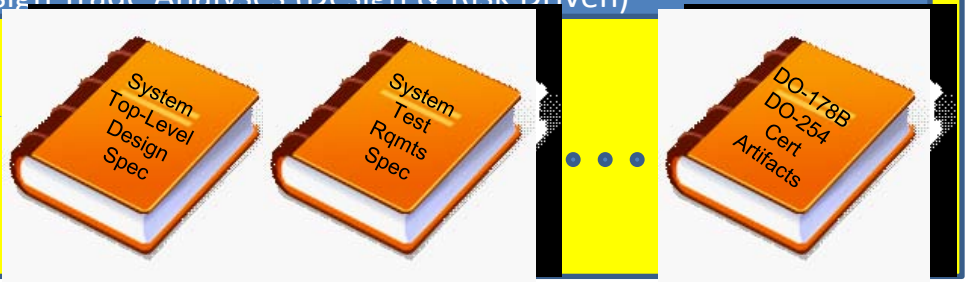
Architectural Candidate 2

Architectural Candidate N

Design Trade Analyses (Design & Risk Driven)

System Operational Environment (Airplatform, et.al.) — System

Down-Selected Functional Architectural Solution



Developer Responsibility

Overview

- Who we are – AMRDEC Software Engineering Directorate (SED)
- Overview of current state of project development with statement of the problem
- Statement of Problem: Some Significant Root Causes
- Solutions to Avoid/Solve the Problem to Achieve Project Success in Minimal Cycle Time
- The process – organization, structure, workflow
- Application & Metrics from two SED projects
- Summary & Conclusion

SED's Model-Based Development Approach

- Our solution to developing a complex system is to employ a model-based approach to develop and mature a system model (simulation) from the earliest point of a project (even during acquisition)
  - No Missile or Aircraft systems will be developed without a simulation
  - Apply the same principals but use a modern, well-supported suite of tools that result in
    - An executable system at every level
    - Highly integrated with other development tools
    - Can auto generate software to support embedded prototype development
- For several projects the IBM Rhapsody tool suite and the associated Harmony SE workflow have functioned as the core around which to develop a system and environment model to achieve our project goals
  - Use this with either UML or SysML
  - Acts as core around which to integrate other products (MATLAB, COTS Graphics Packages, Complex Electronics modeling tools)
  - Supports concurrent test environment modeling and integration
  - Supports real-time development, integration, verification testing
  - Leads right into formal verification testing
- Does NOT replace the enterprise system engineering processes!!

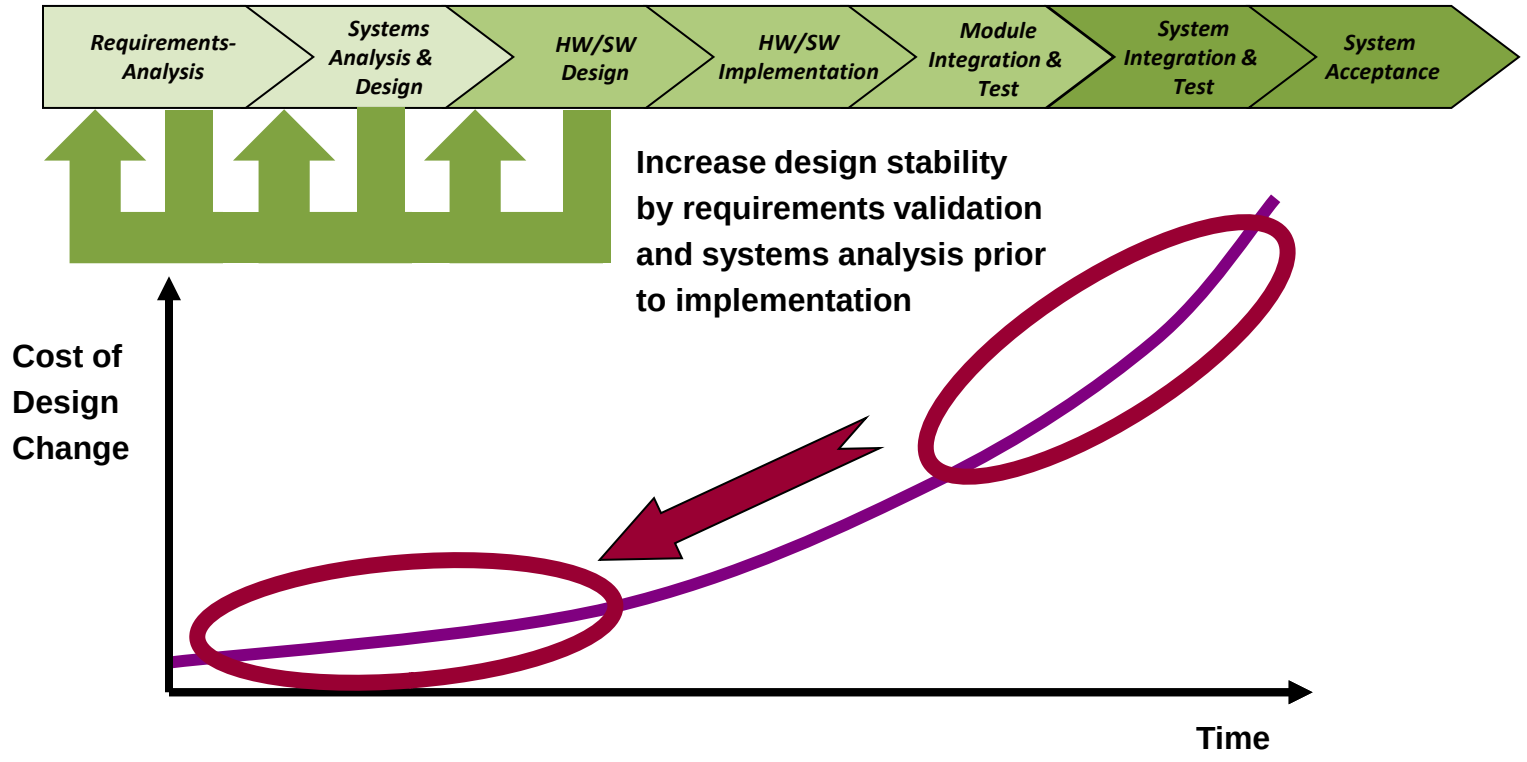
Continuous vs. Discrete Testing

- Continuous vs. Discrete Testing During “Decomposition” Greatly Shortens Formal “Composition” Testing
- Environment/Platform Modeling and Testing begins DURING REQUIREMENTS ANALYSIS!
- Continuous, iterative, model-based testing is conducted as the SUD model is matured.
- The Environment/Platform model (i.e. the SUD Test environment) evolves as the SUD model evolves
- Key Point: Continuous modeling/testing begins before project kickoff, continues throughout the integration/test activities (right side of the “V”).
- This approach differs from the classical “discrete” testing approach, where formal tests are deferred to the right side of the “V”

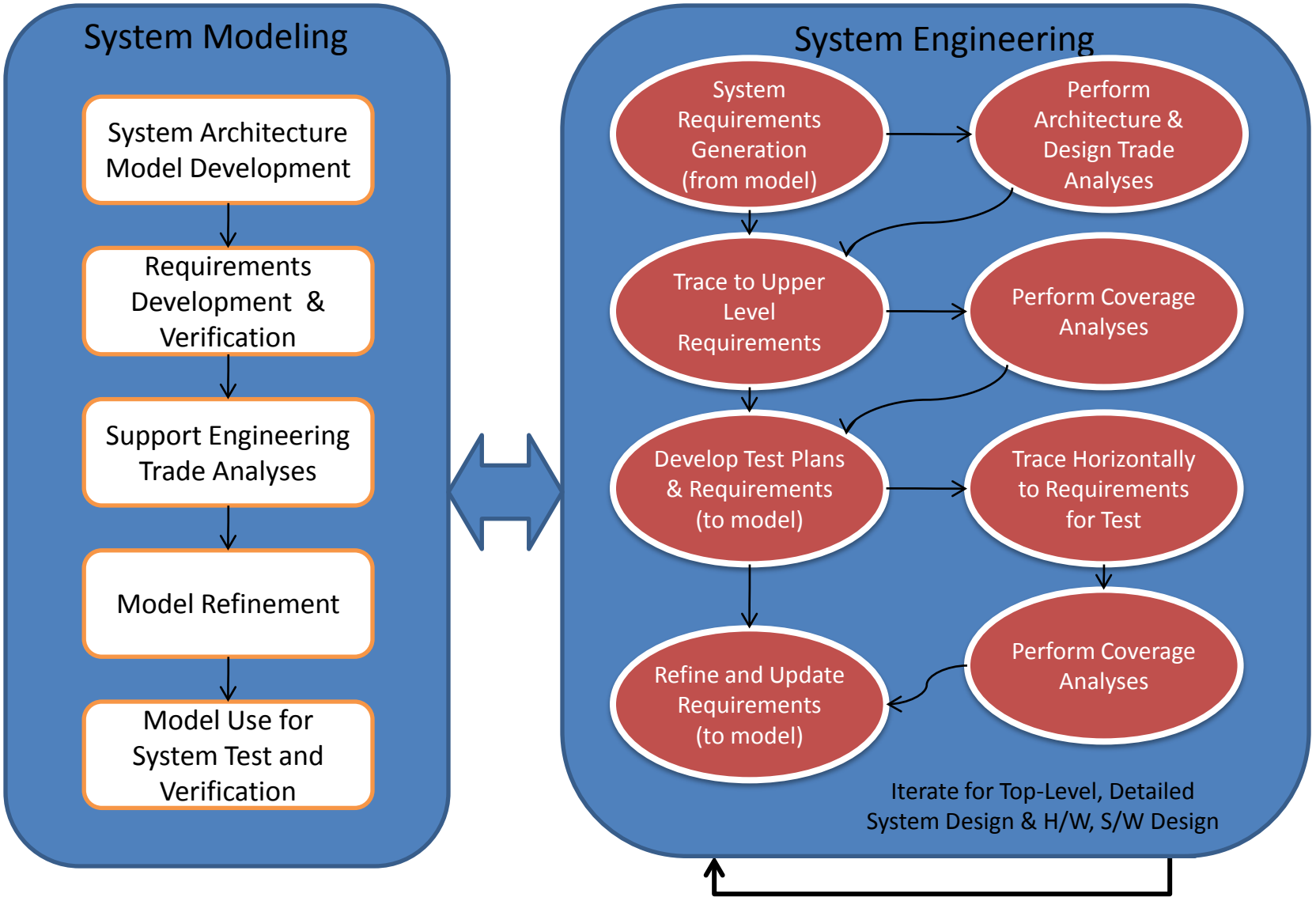
Must mature the test capabilities before starting “composition” formal testing AND eliminate propagation of errors

Model-Based Concurrent Engineering Processes

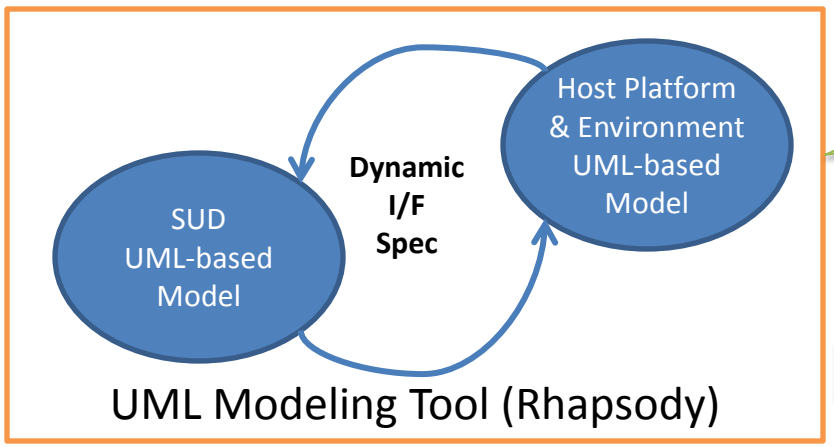
Performing requirements and design verification as early as possible, as opposed to waiting until “composition” activities begin, reduces cost and schedule risks.



Modeling is Used in Conjunction with Standard System Engineering Process NOT Instead of It!

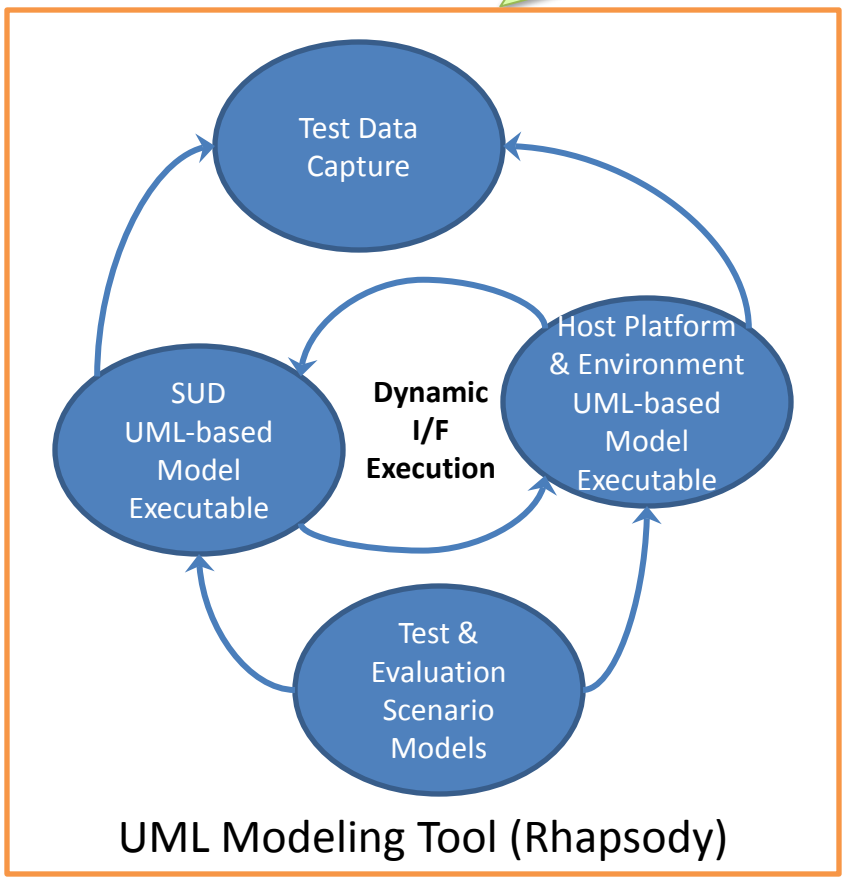


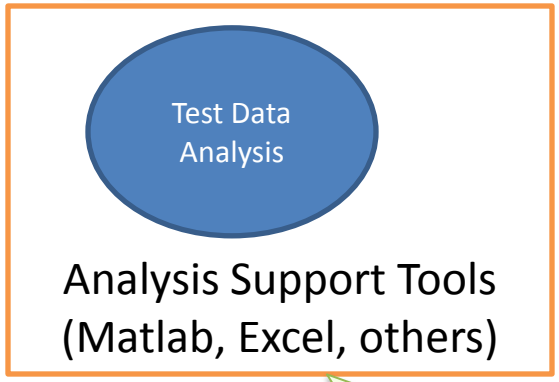
Model Development and Evaluation Activities



Model Development

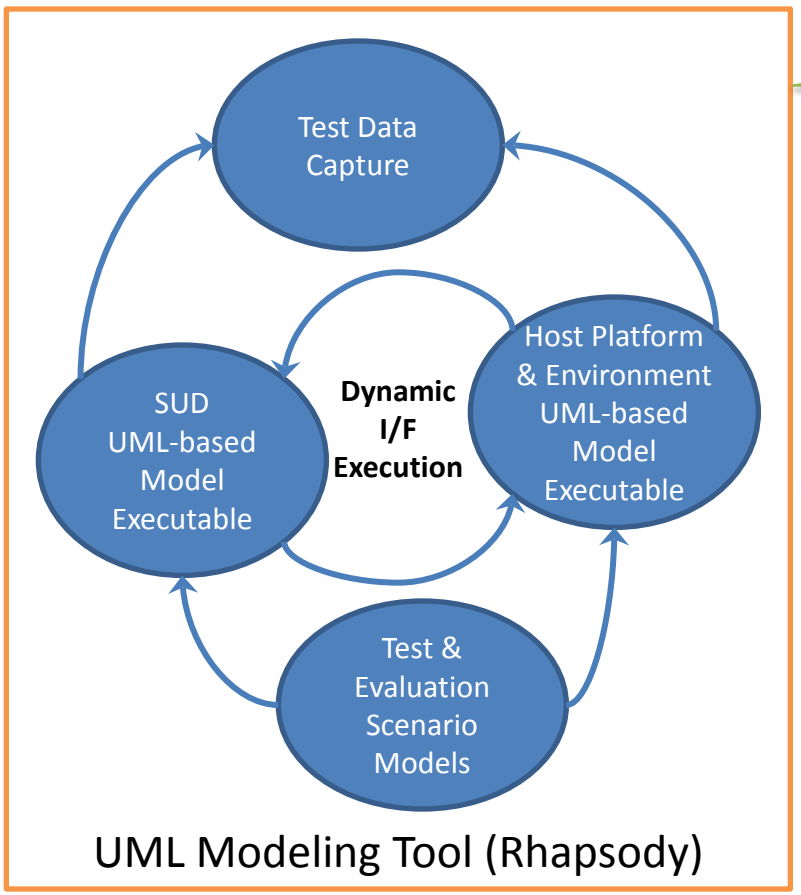
Model Test, Evaluation, & Requirements Pathfinding



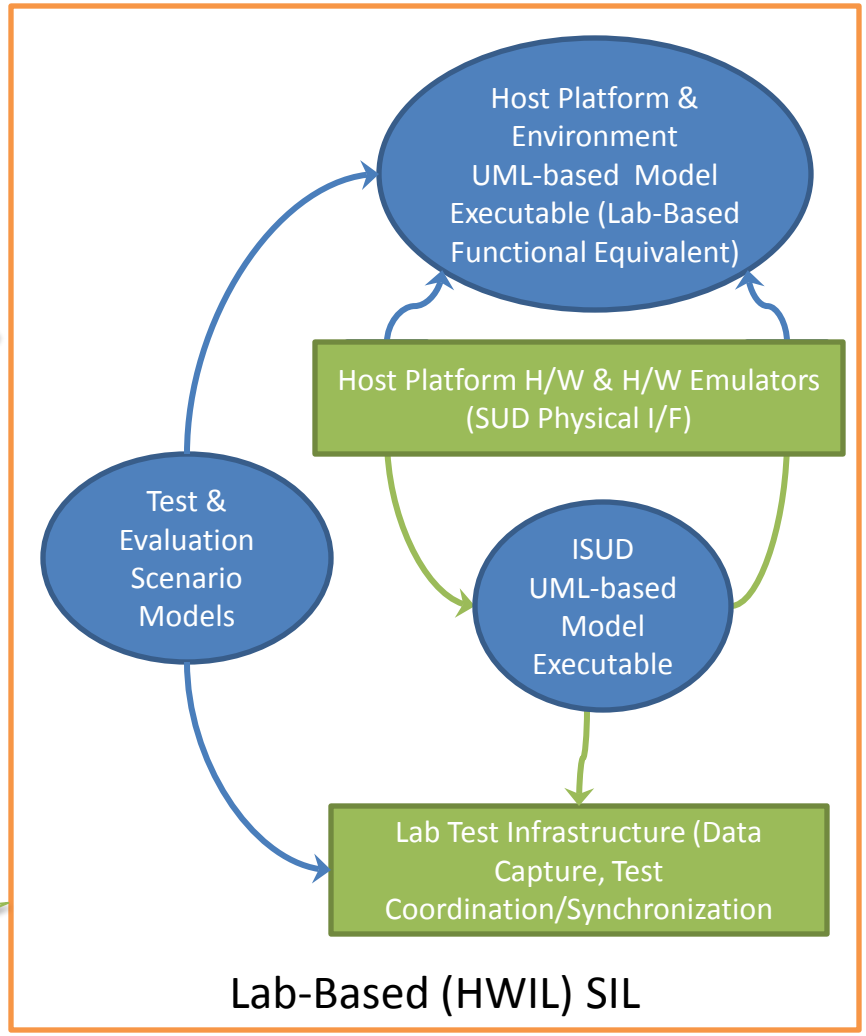


SUD, Environment Model Rqmts Updates/Changes

Model Development and Evaluation Activities Using Lab-Based Test Capabilities



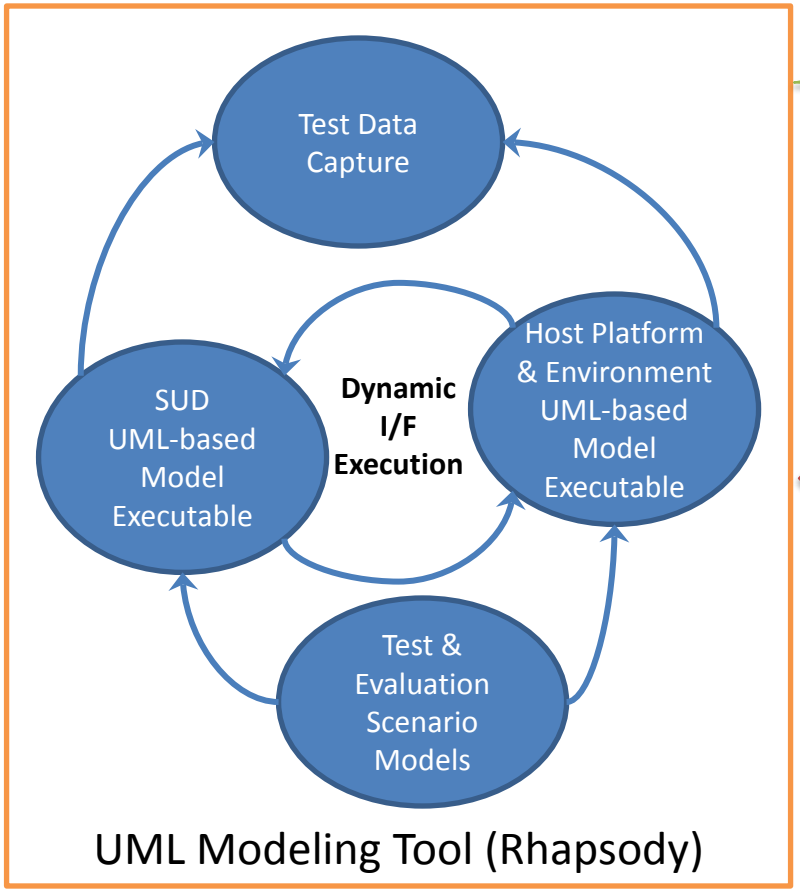
Model Test, Evaluation, & Requirements Pathfinding

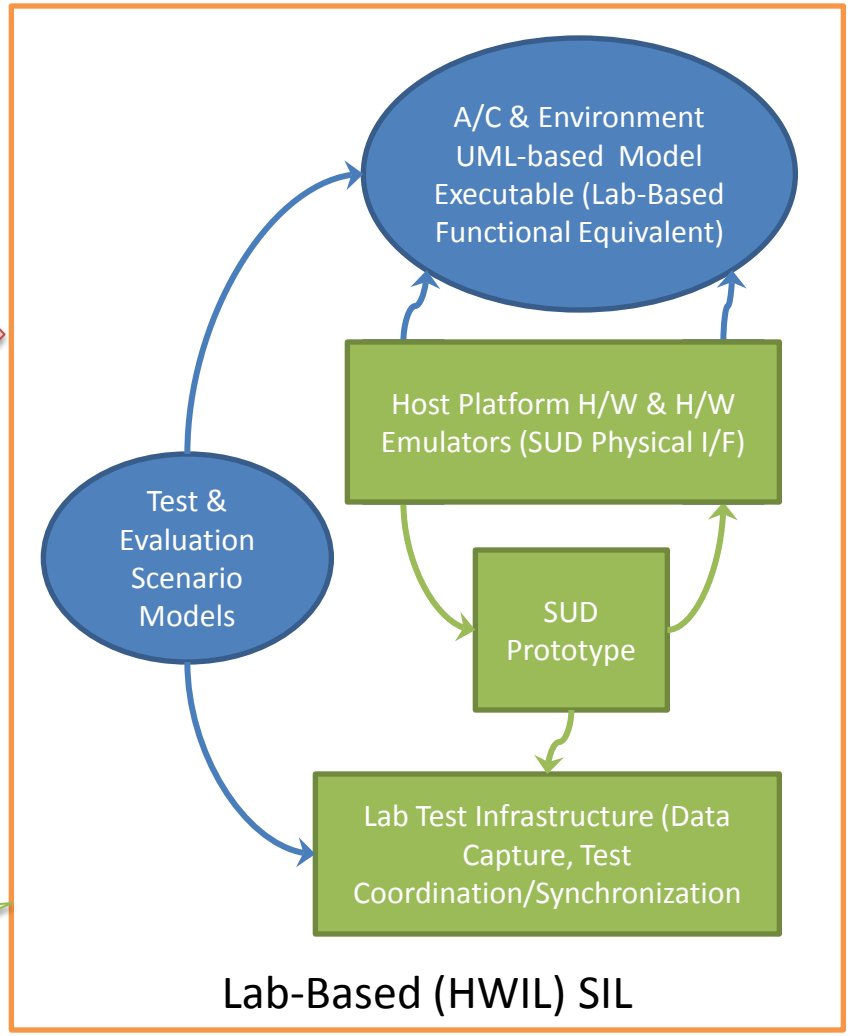


Lab Integration with System Hardware and External Stimulation

Model Development and Evaluation Activities Using Lab-Based Test Capabilities

Model Test, Evaluation, & Requirements Pathfinding





Lab Integration with System Hardware and External Stimulation

Must Utilize Formal, Institutionalized Enterprise Processes (RM, DAR, Design Synthesis)

- Address Risks Early and Continuously
  - Identify risks in *Risk Management Plan* by integrating *Design Trade Analyses* formally into development
  - Perform *Design Trade Analyses* early using the *System Model* using prototypes to explore and mitigate risks
    - Early prototypes should be executed in modeling environment FIRST
    - Build necessary hardware and software prototypes using the model to support maturity of the results
  - Design decisions are supported by formal and rigorous *Decision Analysis and Resolution (DAR)* processes
- Apply an Architecture Design Process that will:
  - Test the seams of your system early and often
  - Eliminate the most expensive defects - between architectural units
- Apply strong architectural modeling techniques
  - Architectural design patterns to reuse best-practice architectures
  - Strong architectures result in adaptable, robust systems

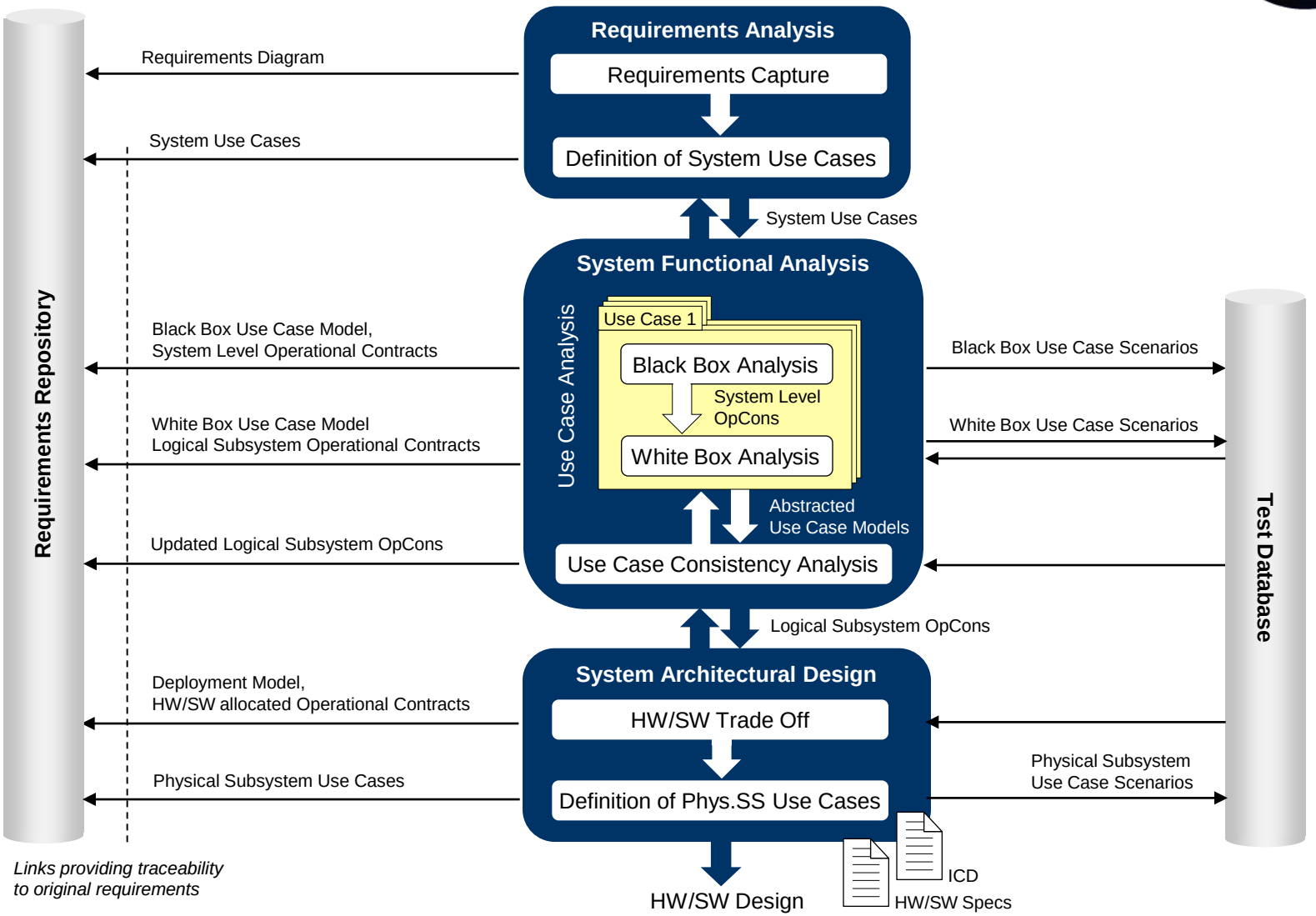
Formalize the Process

- Apply use case-driven development
- Apply a means of deriving design selection
  - For all design activities must use the system MOEs/MOPs -> TPMs/KPPs -> System Parameters in conjunction with design trade analyses.
- Ensure the system completeness and correctness throughout the engineering lifecycle.
- You can only test what you can execute, therefore execute and test early and often.
- Separate logical and physical models - *Reuse* comes largely from redeploying common logical models

Eliminate Costly Labor-Intensive Development Efforts

- Apply Good Tools –
  - Automation as a process improvement strategy can be made quantitatively and economically superior to all of the others.
  - Tools that will automate tasks required for effective Requirements Management, Traceability, Validation, Verification, Implementation and Test. Good tools help support an iterative or spiral process as well as the ability to sustain a system throughout its life.
  - Good Tools are cheap when integrated into enterprise processes , as compared to conventional, manual approaches.
  - For an independent UML 2.0 tool evaluation? Go to: http://www.embeddedforecast.com/REDUML_0304.pdf
- For more information on Process Improvement Strategies go to http://www.dacs.dtic.mil/techs

Example of a Model–Based System Engineering Workflow*



** From the IBM Rational HARMONY® Systems Engineering Workflow*

Overview

- Who we are – AMRDEC Software Engineering Directorate (SED)
- Overview of current state of project development with statement of the problem
- Statement of Problem: Some Significant Root Causes
- Solutions to Avoid/Solve the Problem to Achieve Project Success in Minimal Cycle Time
- The process – organization, structure, workflow
- Application & Metrics from two SED projects
- Summary & Conclusion

Actual Project Characteristics (Project X)

- Army Embedded Communications Device Program
- Work discussed here includes System Specification (SS) Requirements Analysis, Functional Analysis, Functional Decomposition, Development of Functional Test Model and Functional SUD Model (in Rhapsody), Requirements Allocation to Functional Blocks, and generation of a Prime Item Development Spec.
- 45 person effort (includes all engineering, management, and support)
- 6 system architects
- Using Rhapsody & Harmony for Systems Engineering (SE) workflow
- Project is ongoing

Project X Requirements Sources

- **Created System Specification from Legacy Documents**
  - 47 legacy use case docs
  - Legacy hardware spec
  - 2 Legacy SW Specs
  - Platform Implementer's Guide
  - Legacy SW Code
  - SOW
  - Emails

Project X Work Products Generated to Date (Project is On-Going)

- Prime Item Development Spec (PIDS) From System Spec
- Functional Analysis
  - 7 Use Cases (from original “47 use cases”)
  - 6 System Architects Developed Independent UC Models (~ 3 Month Effort)
  - 350 Derived Requirements Discovered (captured in PIDS)
  - 50 SS Requirements Holes Identified during functional model simulation analysis
- Derived & SS Requirements vetted &captured in DOORS
- PIDS (top-level design) Document Generated from DOORS
- Successful System Functional Review

Project X Continuous Test Approach Identified Errors During Design

- The Continuous Test Approach Identified Errors During Requirements/Functional Analysis Phase of Development
- Modeling the Test Environment concurrently with, and independently from, requirements/systems model, and executing them against each other, uncovers interface discrepancies and identifies uncovered requirements early (during requirements development phase).
  - A significant interface issue was discovered and corrected DURING REQUIREMENTS ANALYSIS phase via testing the FUNCTIONAL Rhapsody model using the FUNCTIONAL Test Environment Rhapsody model.
  - A significant requirements discrepancy was discovered via FUNCTIONAL SUD Rhapsody Use Case model simulation.
- Development and Continuous Execution of Test Models Concurrently with SUD Model Development Reduces Error Propagation while it is still "Cheap to Fix the Errors"
- Test Environment Modeling must begin ASAP, and must be matured with the system (SUD) and its environment models.

Project X Lessons Learned

- Concurrent development and implementation of the Test Environment model saves time by identifying errors before they can be propagated.
- Error propagation is mitigated early, (even during requirements analysis) using concurrent, Model Based Testing to drive SUD model
- Harmony work flow standardizes work products
- Don't attempt this without training
  - Even with training, continued mentoring is vital
  - Training is necessary but not sufficient
  - This approach may not be cost effective if it is not institutionalized (cost may be prohibitive if only used on one project)
- Must integrate model-based development activities into standard enterprise system engineering – Rhapsody Harmony doesn't replace system engineering processes.
- Independent SUD functional model development per use case followed by integration of models is labor intensive
- Rhapsody Harmony SE wizards provide significant productivity increases, but…
  - If you don't understand what the wizards do… trouble
- Time is saved when transitioning from Systems Engineering to SW Engineering due to a common modeling tool suite (Rhapsody) and language (SysML & UML)

Actual Project Characteristics (Project Y)

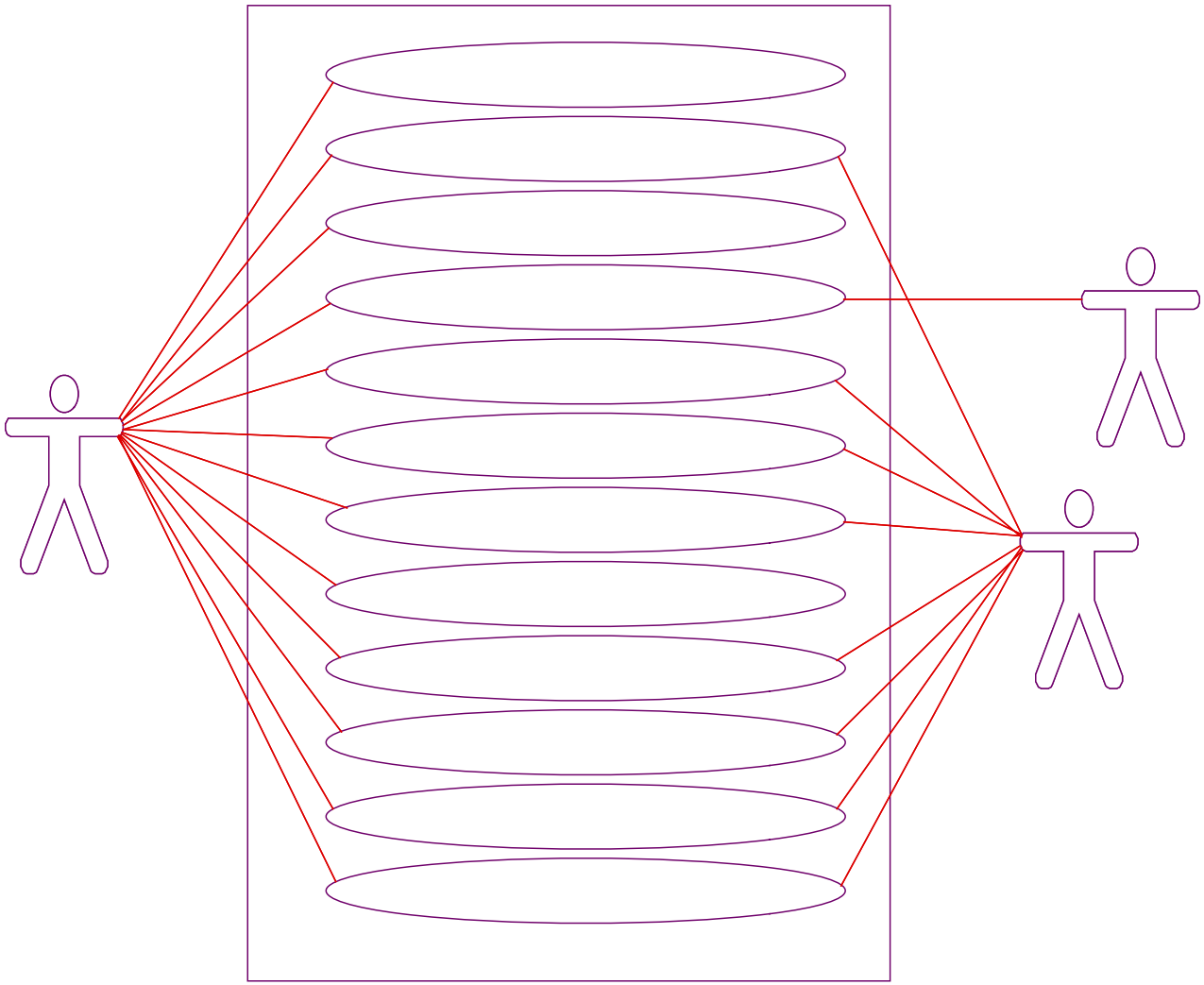
- An embedded logistics/RAM fielded support device
  - System is to be developed for an initial system
  - Then adapted to support all Army-related systems
  - Need good architecture
    - Life cycle cost optimization is essential
    - System will be used and adapted for new systems for many years
    - Initial development is two-year effort with three system builds
    - Currently working on readying system Build 0
  - Build 1 schedule is EXTREMELY aggressive – only a one year development to early fielding/deployment with full security/IA certification!
  - Couldn't get there without support of automated development tools during design, verification, certification

Actual Project Status/Metrics (Project Y)

- Model development proceeding as planned with good results
  - Have identified twelve Use Cases (see next chart)
  - Have reviewed and vetted primary Use Cases (for initial project system build) with stakeholder
  - Have identified all nominal and off-nominal scenarios and created sequence diagrams and state transition diagrams
  - Implemented all data structures defined in the system ICD via data object and message class definitions
  - Have defined object oriented and fully abstracted physical architecture (non-functional requirements) and they are now implemented in the model along with the functional requirements
  - Performance requirements will be forthcoming
  - Currently have executable model representing 15K SLOC (debug/animated version) and 10K SLOC release version from approximately 200 man-hours total team expenditure on model effort
  - Already vetting our source code with the IA team with good results

SO FAR – SO GOOD!!!!

Project Y System Use Case Diagram

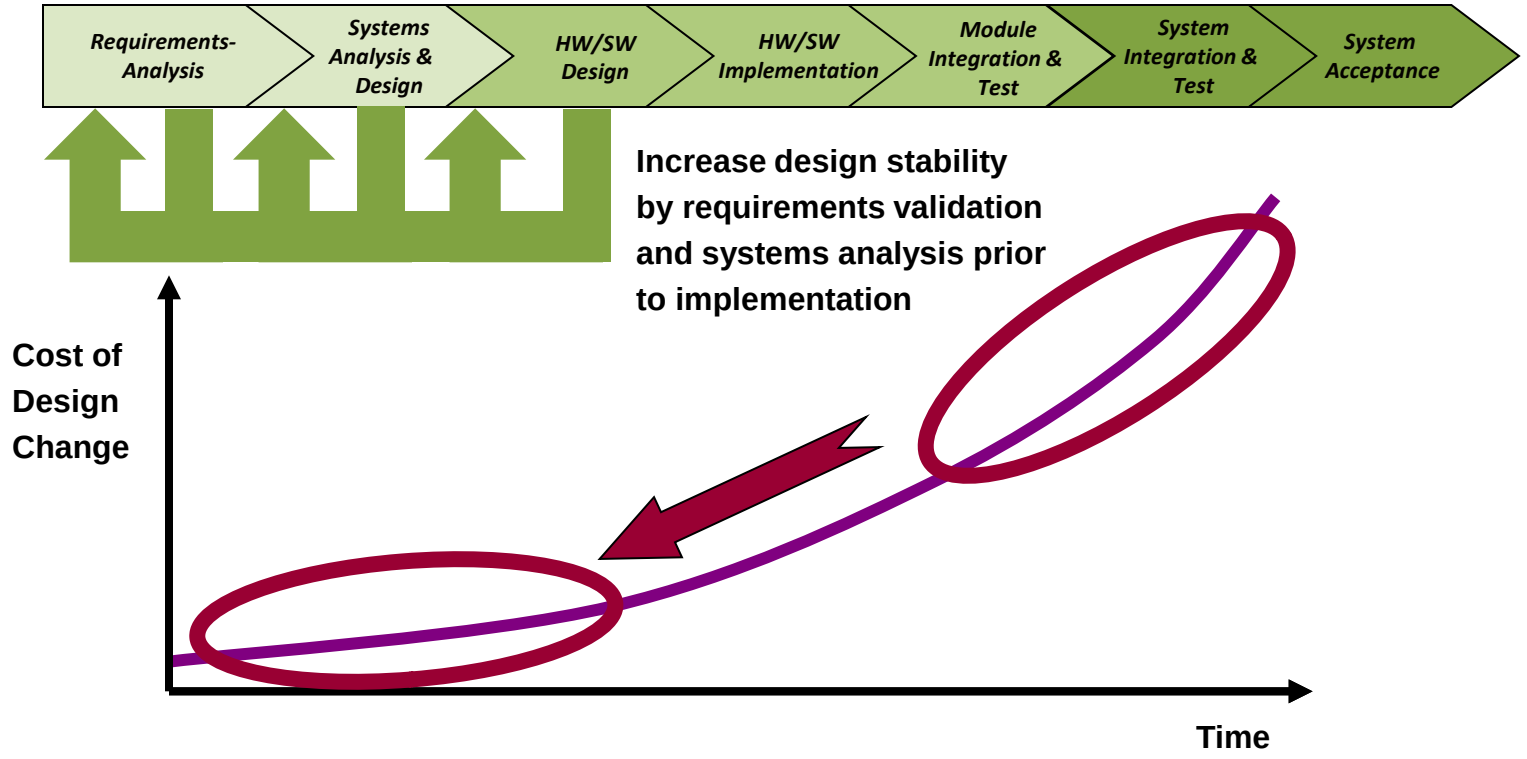


Overview

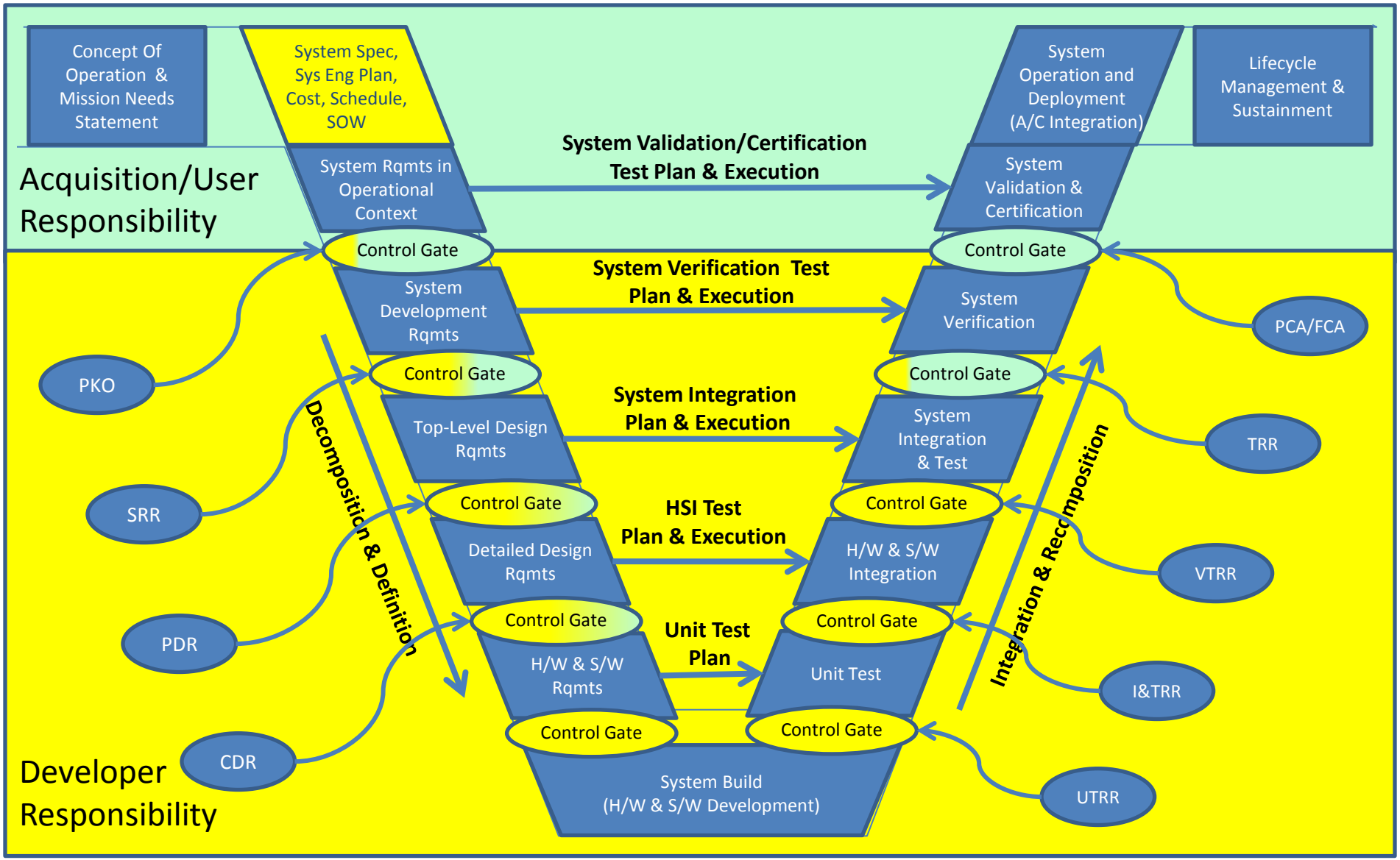
- Who we are – AMRDEC Software Engineering Directorate (SED)
- Overview of current state of project development with statement of the problem
- Statement of Problem: Some Significant Root Causes
- Solutions to Avoid/Solve the Problem to Achieve Project Success in Minimal Cycle Time
- The process – organization, structure, workflow
- Application & Metrics from two SED projects
- Summary & Conclusion

Model-Based Concurrent Engineering Processes

Performing requirements and design verification as early as possible, as opposed to waiting until “composition” activities begin, reduces cost and schedule risks.



Integrated Model-Based Development Overview



Summary and Conclusion

- Must have formalized enterprise processes for system development
  - Must use a strong Integrated Product Development (IPD) team
  - Must have engineering team working together from the beginning of the project
  - Involve the specialty engineering team elements early (Safety, RAM, Security/IA, T&E, CM, QA/QC, etc)
- Must supplement system engineering capabilities and processes with a suite of integrated tools to provide automation
  - Must use a variety of general and specialized tools for each unique attribute of the system
  - Must integrate the tools with a core modeling tool supported by an industry standard language (UML, SYSML)
  - The core modeling tool must support integration and interoperability of supporting models such as MATLAB Simulink, STK, Complex Electronics, Architecture (AADL)
  - We have integrated Rhapsody and the Harmony SE workflow into our enterprise processes and used them to good effect to meet project needs.

Summary and Conclusion (Continued)

- Must use rigor and focus to eliminate the Six Root Causes of development problems and schedule killers
- Must use a Model-Based Development approach to develop a fully functional model (simulation) of the system and its operational environment
  - And use the model to support formal RM, DAR, Engineering Design Trade Analyses
  - Must ensure design quality integrating system MOEs/MOPs into the design in a formal and rigorous manner
- Must utilize the System and Environment Model to integrate, verify, mature the system suite of integration and verification test capabilities BEFORE starting the formal Composition activities on the backside of the “V”


RDECOM
AMRDEC
SED
Software Engineering Directorate


AMRDEC
STRENGTH THROUGH TECHNOLOGY

Questions?