Software Reviews Since Acquisition Reform – Architecture-Driven Considerations

Dr. Peter Hantos

Senior Engineering Specialist

Software Acquisition and Process Office

Ground Systems Architecture Workshop 2004
ACE2 Breakout-Group Session

THE AEROSPACE CORPORATION

Acknowledgements

- This work would not have been possible without assistance from the following:
  - Reviewers
    - Richard J. Adams, Software Acquisition and Process Office
    - Suellen Eslinger, Software Acquisition and Process Office
    - Karen L. Owens, Software Acquisition and Process Office
    - Mary A. Rich, Principal Director, Software Engineering Subdivision
  - Sponsor
    - Michael Zambrana, USAF Space and Missile Systems Center, Directorate of Systems Engineering
  - Funding source
    - Mission-Oriented Investigation and Experimentation (MOIE) Research Program (Software Acquisition Task)

Agenda

- Perspectives on Review Issues
- Architecture-Driven Considerations vs. Architecture Reviews
- The Computer Software Configuration Item Controversy
- Functional Decomposition
- Architectural Layer Dependencies
- Use Cases
- Components of Implementation
- Technical Performance Measurements
- Conclusions

Background of Problem

- **Pre-1994:**
  - MIL-STD-1521B (Technical Reviews)
    - Formal milestone reviews
    - Date of last version is June 4, 1985 (!)
    - Supporting DoD-STD-2167A (Defense System Software Development)
- **1994:**
  - MIL-STD-498 (Software Development & Documentation)
    - Although all other MIL standards are cancelled by the DoD, MIL-STD-498 was approved as an interim standard for 2 years
    - Joint reviews: Schedule and content proposed by contractor
- **Now:**
  - **No official development or review standards of record**
    - Each acquisition defines a minimum set of major contractual technical reviews and associated entrance/exit criteria in its Integrated Master Plan, nevertheless:
      - Neither the government nor the contractor has a clear concept of what reviews should contain and when they should occur
      - Interpretation of those reviews (e.g., System PDR, System CDR) is left to individuals to decide
    - Quality and content of reviews is widely different both within and across programs
    - Quick, last-minute, before-review efforts to revive and customize MIL-STD-1521B proved to be ineffective

Perspectives on Review Issues

- **The Life Cycle Perspective ("When?")***
  - Pre-acquisition reform assumptions:
    - Acquisition and development are exclusively Waterfall
    - Reviews (SSR, PDR, CDR, etc.) are clearly positioned
  - Now:
    - Evolutionary Acquisition
    - Iterative/Incremental and Spiral Development
    - Emerging agile methods
    - Asynchronous, in-process, interim reviews

**** For more details see my upcoming presentation at the 2004 Systems & Software Technology Conference in Salt Lake City, Utah: Hantos, P.,"Software Reviews Since Acquisition Reform – The Life Cycle Perspective"***

Perspectives on Review Issues (Cont.)

- **The Artifact Perspective ("What?") ***
  - Evolving performance and maturity of process artifacts and work products along the development life cycle
  - Key areas of interest in analyzing the impact of new software development trends:
    - Architecture
    - Product-oriented software engineering activities
    - Engineering management processes
    - Integral software engineering activities
    - Hardware-software technology
    - Security

**** For more details see my presentation at the Third Annual Conference on the Acquisition of Software-Intensive Systems in Arlington, VA: Hantos, P.,"Software Reviews Since Acquisition Reform – The Artifact Perspective", January 26-28, 2004***

Presentation Objectives in the Context of Workshop Objectives

- **Emphasize** the importance of architecture as a basis for
  - Understandability
  - Assessing maintainability, extensibility, and executability
- **Clarify** the difference between reviewing architecture vs. architecture-driven considerations during technical reviews.
- **Demonstrate** the inadequacy of **MIL-STD-1521B** as the basis for design reviews on architectural grounds
  - Show the flaws in the Configuration Item (CI) concept
  - Discuss requirements traceability and verification of the completeness of the design
  - Discuss specification and review of Technical Performance Measurements (TPMs)
- **Non-objective**: How to conduct Architecture Reviews

Architecture-Driven Considerations vs. Architecture Reviews

- **Architecture review objectives:**
  - Present business drivers underlying the architecture
  - Present the definition of the system/software architecture
    - Structure, behavior, collaboration, constraints, and quality attributes of components and interfaces
  - Concentrate on significant elements that have a wide impact on:
    - Structure, performance, robustness, evolvability, and scalability
  - Present considered architectural approaches and decision rationale
  - Present architecture style choices and decision rationale
  - Present architecture evolution parameters
  - Demonstrate consistency among:
    - Concept of Operations (CONOP)
    - Developed prototypes
    - Requirements
    - Architecture

Architecture-Driven Considerations

- **During the review of the design, be cognizant of the underlying architecture-centric development process:**
  - Major architectural decisions:
    - Ensure that the design does not conflict with major architectural decisions
  - Architecture is more than a static blueprint:
    - It is dynamically **built**, **validated**, **baselined** and **elaborated** during the iterative/incremental development life cycle
  - Architectural views and related models*:
    - **Design** model – Logical view (Top level abstractions)
    - **Process** model – Process view (Logical view for complex systems)
    - **Implementation** model – Implementation view (Organization of modules)
    - **Deployment** model – Deployment view (Mapping runtime components)
    - **Use-case** model – Use-case view (Validating the integrity of different views)

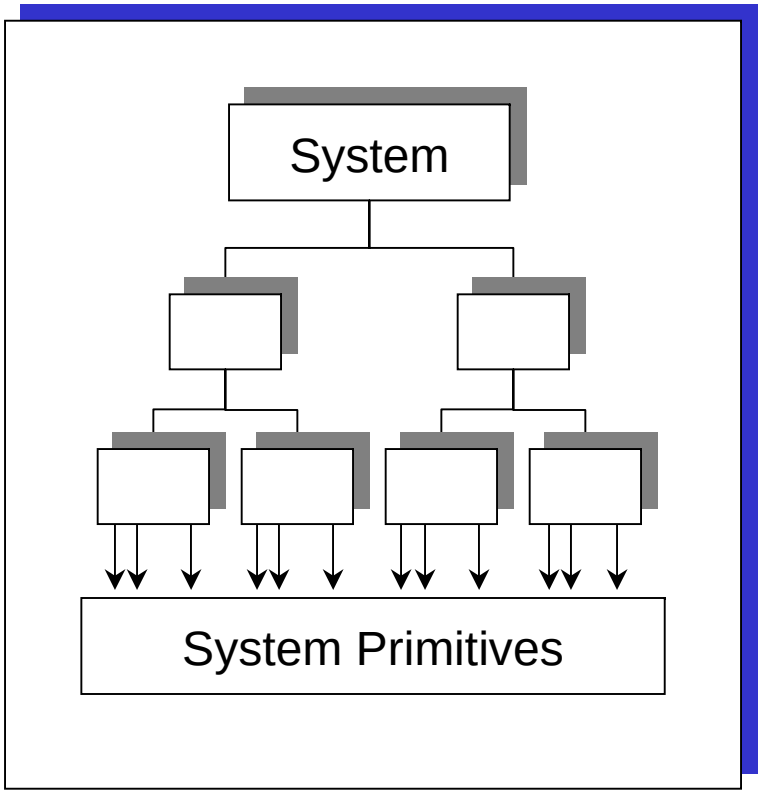
** Definitions of architectural views are from Reference [4]*

The CSCI Controversy

- **Current software development methods are not driven by acquisition standards:**
  - The term “CSCI” is counter-intuitive*, since even lower level elements of the system must be under Configuration Management
- **Object-Oriented (OO) concepts and terminology are the norm:**
  - **Objects** – with flexible granularity
  - **Packages** – for depicting logical object structure
  - Deployment of **Components** on **Nodes**
    - **not CSCIs on HWCIs**
  - Distinction between **source code** and **executable files**
  - Multiple, **dynamic object instantiation,** use of **O**bject **R**equest **B**rokers
  - **Dynamic linking**
  - Distinction between **Analysis**, **Design** and **Implementation**

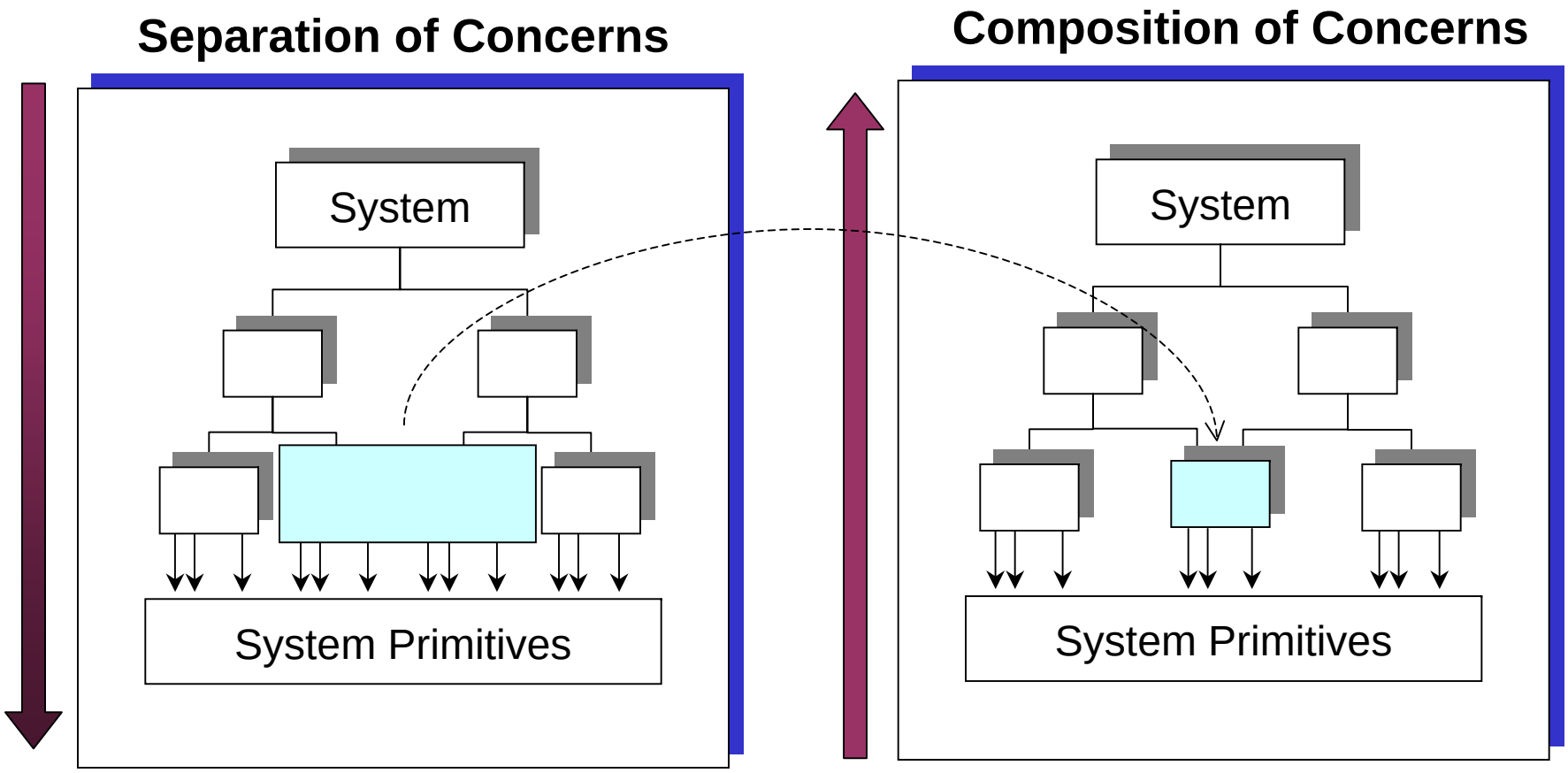
** For the definition of a Configuration Item based on DOD-STD-480 please see backup slide #24*

Functional Decomposition



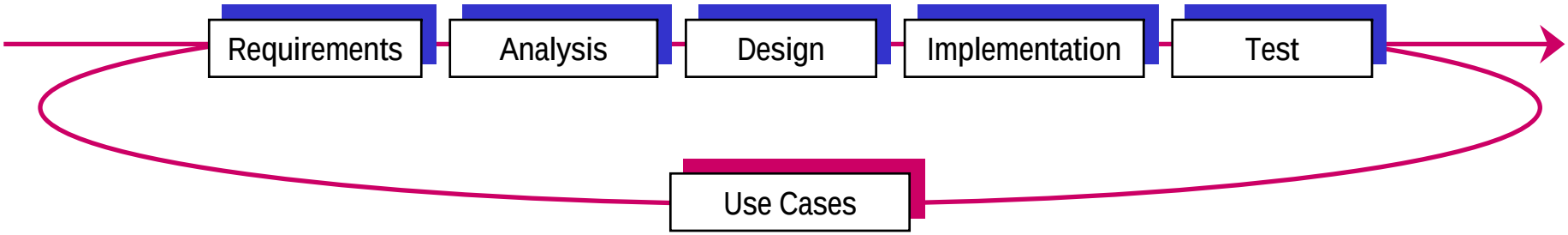
- Successively decomposed the system to system primitives level
- Supposed to provide requirements traceability to validate completeness of the design, but
- … No guarantee that these primitives will work together on higher levels
- … Does not deal with non-functional requirements (performance, quality, security, etc.)
- The architecture should have been evolving during decomposition

Synthesis – An Iterative Analysis/Design Circle



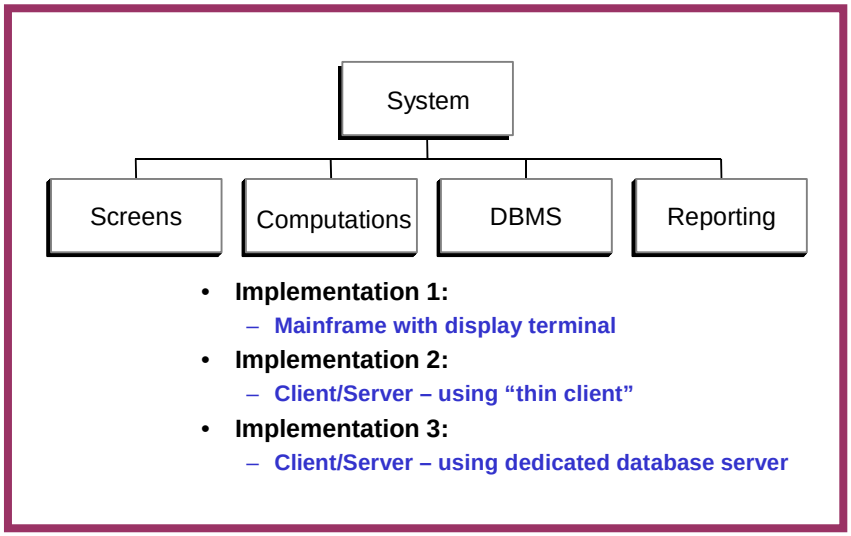
** For a less abstract JAVA implementation example please see backup slide #25*

OO: The Case for Use Cases

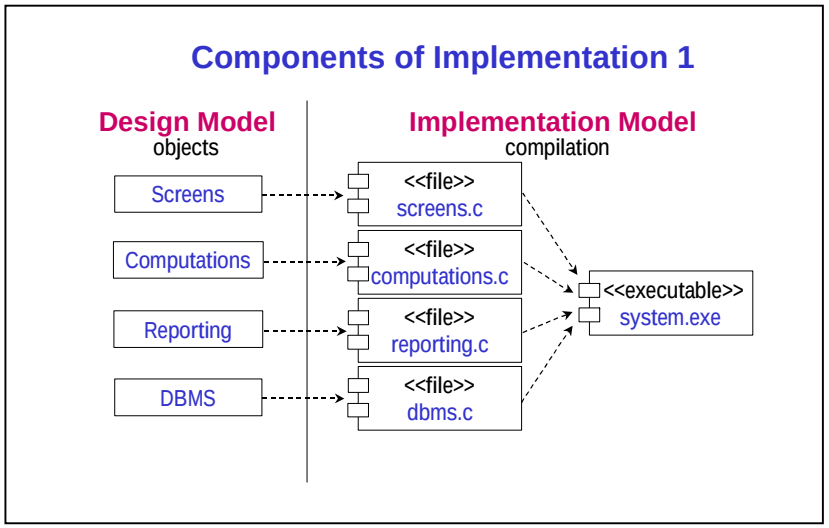


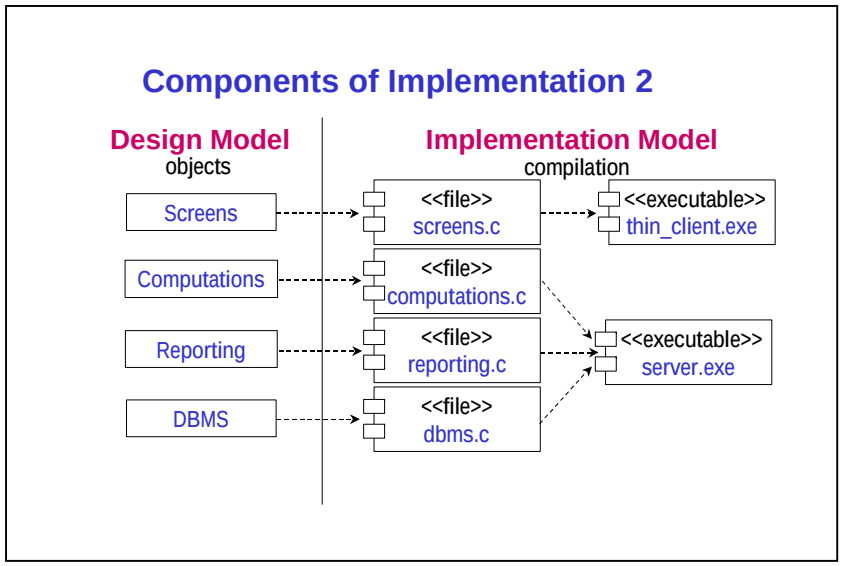
- **Use Cases are not just for capturing requirements:**
  - Use Cases bind the core workflows
  - Each development increment is a working realization of a set of Use Cases
  - Multi-level hierarchy of Use Cases:
    - **Top level:**
      - **External Use Cases** – **System behavior and actors**
        - » **Caveat: Use Cases cover only functional requirements**
    - **Multiple lower levels:**
      - **Internal Use Cases** – **Subsystem behavior and relationships**

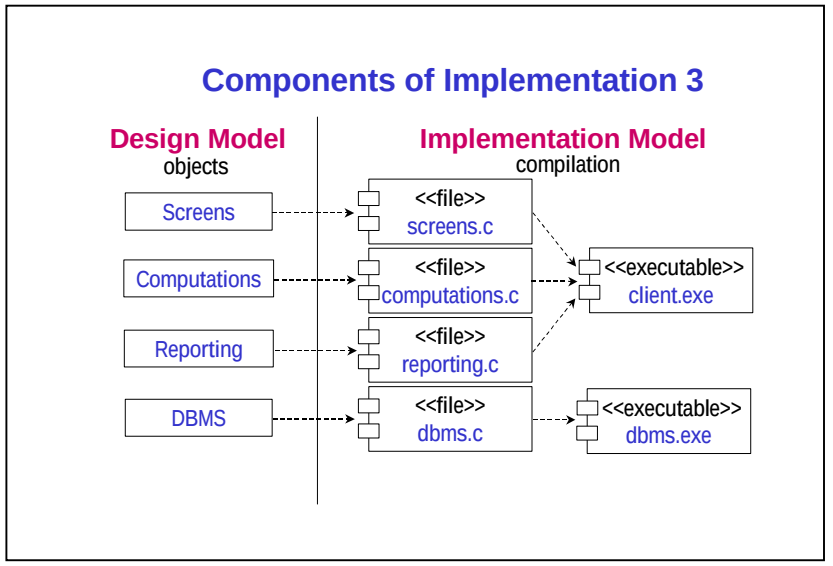
Components of Implementation



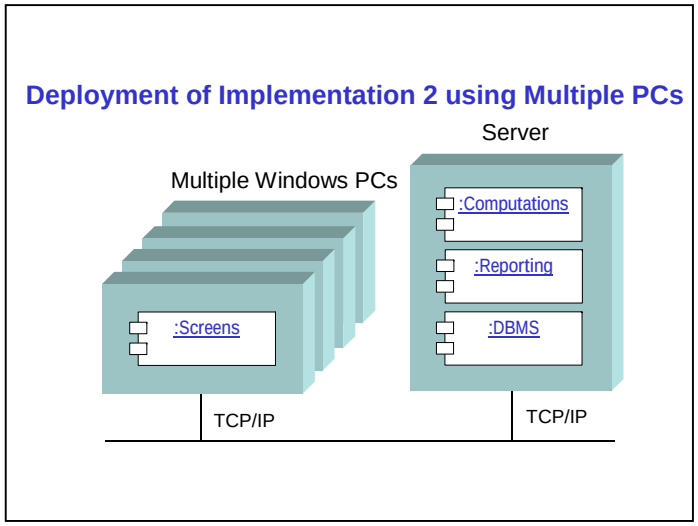
- **Implementation 1:**
  - **Mainframe with display terminal**
- **Implementation 2:**
  - **Client/Server – using "thin client"**
- **Implementation 3:**
  - **Client/Server – using dedicated database server**

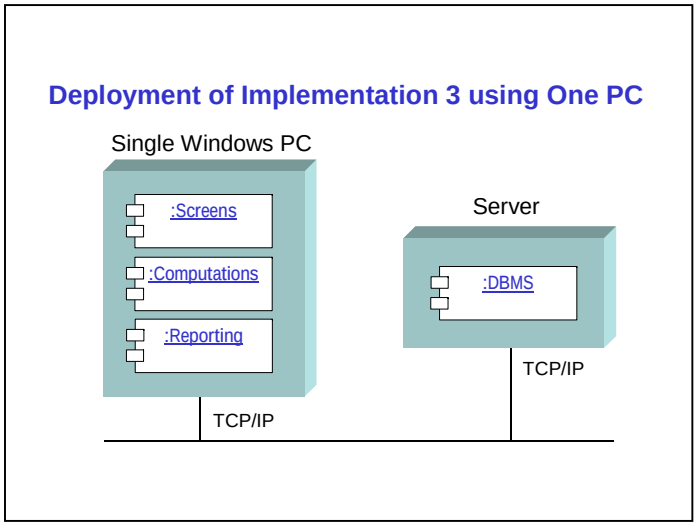


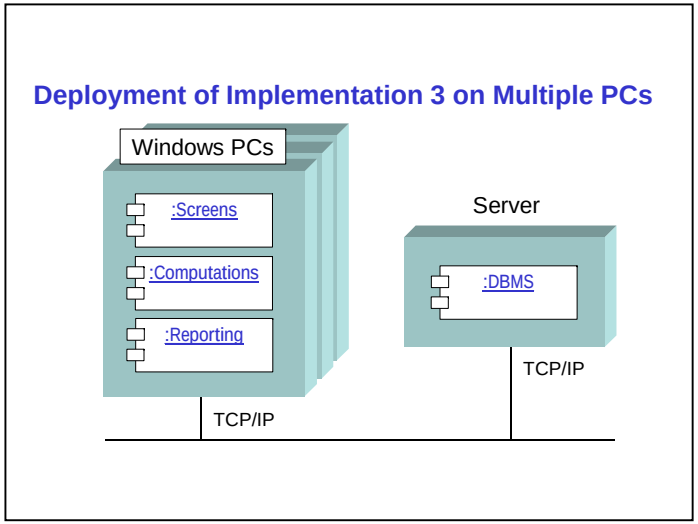




Technical Performance Measurements







Sample TPM's:

- Operator response time to commands < 3 sec
- Results of computation are presented in 20 sec

System-level TPMs:

- Have to be allocated to nodes

Reviews have to track the allocated TPMs:

- The end-to-end, system-level TPM includes performance on the node *and* on the network

Conclusions

- Architecture-driven considerations are essential in carrying out successful software technical reviews
- MIL-STD-1521B is inadequate as the basis for design reviews
- Understanding of object-oriented methodologies is critical in planning software technical reviews
- Functional Decomposition will not provide Requirements Traceability
- In-process reviews must track allocated TPMs
- The **C**onfiguration **I**tem concept is not supportive of modern development practices

Acronyms and Abbreviations

CDR	Critical Design Review
CI	Configuration Item
CONOP	Concept of Operations
COTS	Commercial Off-the-Shelf
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
DBMS	Database Management System
DoD	Department of Defense
HWCI	Hardware Configuration Item
MIL	Military
OO	Object-Oriented
ORB	Object Request Broker
PC	Personal Computer
PDR	Preliminary Design Review
SCM	Software Configuration Management
SSR	Software Specification Review
STD	Standard
TCP/IP	Transmission Control Protocol/Internet Protocol
TPM	Technical Performance Measurements

References

[1] Booch, G., Rumbaugh, J., and Jacobson, I., “The Unified Modeling Language User Guide”, Addison Wesley Longman, Inc., 1999

[2] Barry, B., “Spiral Development: Experience, Principles, and Refinements”, CMU/SEI-2000-SR-008

[3] Clements, P. C., “Active Reviews for Intermediate Designs”, CMU/SEI-2000-TN-009

[4] Kruchten, P., “The Rational Unified Process – An Introduction”, Addison Wesley Longman, Inc., 1998

[5] Smith, J., “The Estimation of Effort Based on Use Cases”, www. rational.com, Rational Software, 1999

Contact Information

Peter Hantos
The Aerospace Corporation
P.O. Box 92957-M1/112
Los Angeles, CA 90009-2957
Phone: (310) 336-1802
Email: peter.hantos@aero.org

Backup Slides

Refresher: MIL-STD-1521B Characteristics

- Assumes Waterfall Development Model
- Assumes Functional Decomposition of Requirements
- Top-level system building blocks are configuration items:
  - **HWCI** – Hardware Configuration Item
  - **CSCI** – Computer Software Configuration Item
- High-Level Design == Architecture
- Hardware-software development processes are clearly separated:
  - Design trade-offs only on systems engineering level
- Clearly positioned, formal milestone reviews

Computer Software Configuration Items

- **Definition of CSCIs (DOD-STD-480):**
  - SW that is designated by the contracting agency for configuration management
- **Structural Breakdown of CSCIs:**
  - CSC (**C**omputer **S**oftware **C**omponents)
    - CSU (**C**omputer **S**oftware **U**nits)
- **Granularity of CSCIs:**
  - Coarse, typically 7-8 CSCIs even in large systems
- **Static View of Software:**
  - Assumes unchanging configuration entities across the life cycle:

    Design → Source Code → Developmental Executables → Delivered Executables

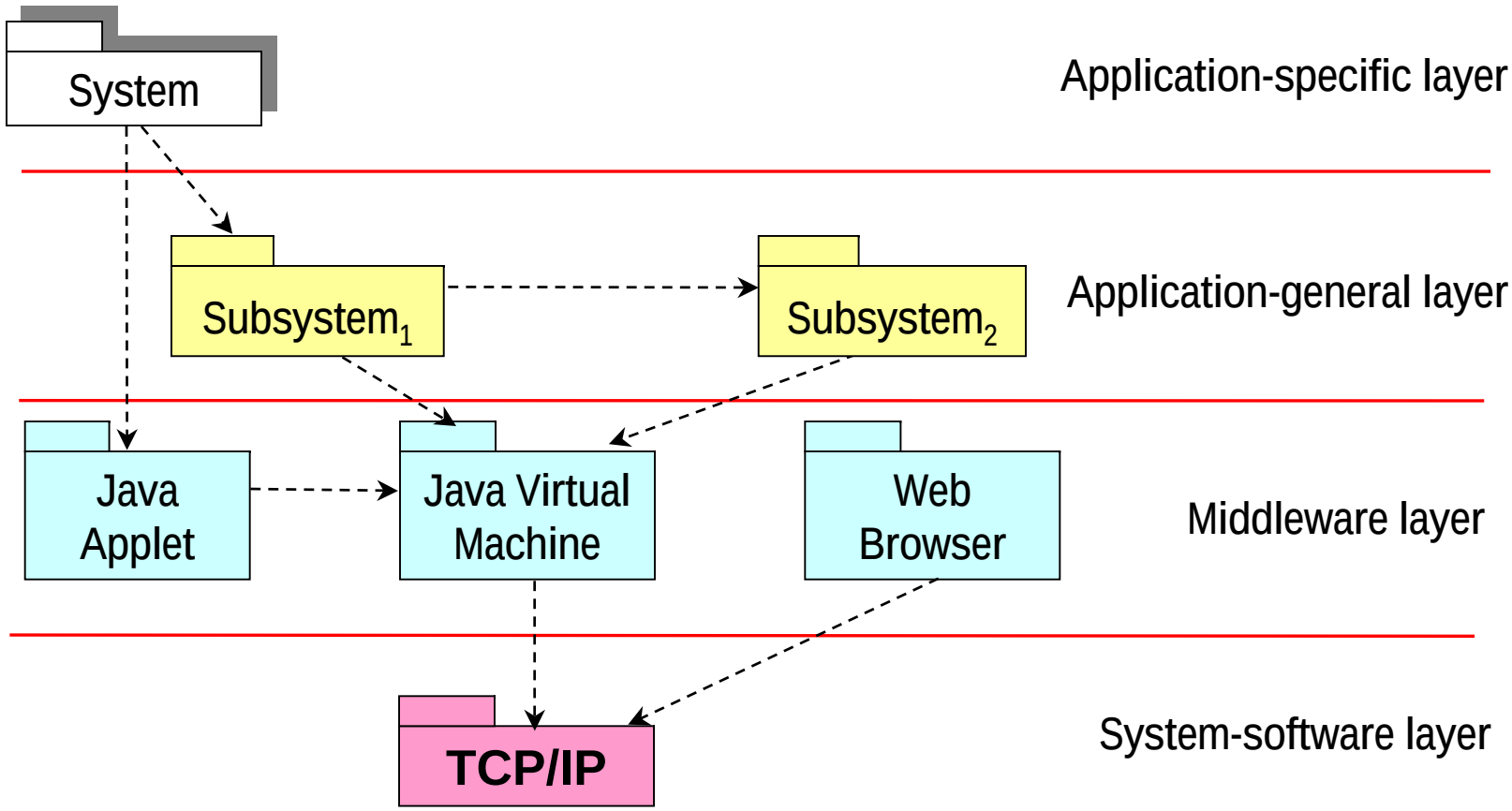
More on Functional Decomposition

- **Functional Decomposition is misused:**
  - Its purpose is to understand and communicate system requirements and **NOT** synthesis (allocation of sub-functions to solution structures)
  - It doesn't provide complete Requirements Traceability
- **Functional Decomposition has to be a multi-level process:**
  - On each level we describe the required behavior, and
  - Trying to find a solution to implement it before deciding whether the behavior on the next level needs to be refined
- **In OO the equivalent activity is Analysis**
- **In OO functional requirements are documented via Use Cases**

Analysis is NOT Design or Implementation

- **Analysis:**
  - "Rough sketch" of the system
  - Description of the system in the application domain
  - Helps us to refine and understand functional requirements
  - Lets us reason about the internals and internal resources
- **Design:**
  - Allow us to shape the architecture that satisfies all, functional *and* non-functional requirements
  - Design is a refinement of analysis
- **Implementation:**
  - Creation of the system in terms of components, such as source code, scripts, binaries, executables, etc.
  - Component testing
  - System integration
- **All three have to be considered for complete Requirements Traceability**

Architectural Layer Dependencies Example



Note that the topology of the architecture is not a tree-structure anymore