Software Acquisition Management

TOPIC LEARNING OBJECTIVES

Upon successful completion of this topic, the student will be able to:

1. Identify common ways that software-intensive projects have gotten into trouble.
2. Identify "Best Practices" that may be appropriate for the acquisition of software-intensive systems.
3. Relate the typical distribution of software life-cycle costs to the planning of an acquisition program.
4. Identify key solicitation planning activities that should be done when acquiring a software-intensive system.
5. Select an appropriate approach (e.g., analogy, parametric, top-down, bottom up, Delphi) to estimate the cost and schedule for a software-intensive system.
6. Identify key life-cycle planning issues in developing an acquisition strategy for a software-intensive system.
7. Identify typical software development life-cycle activities and standards.
8. Recognize the relationship between software development activities and the system engineering process.
9. Recognize how solid requirements analysis and comprehensive testing are the marks of an effective software development program.
10. Recognize the software development models.
11. Given a software-intensive system (such as a telecommunications or guidance system), select an appropriate software development methodology.

STUDENT PREPARATION

Student Support Material

1. None

Primary References

1. Defense Acquisition Guidebook, Chapter 3
2. MIL-STD-498 Software Development and Documentation

Additional References

1. DAU ISA 1011 Basic Information Systems Acquisition
2. DAU ISA 201 Intermediate Information Systems Acquisition
3. DAU CLB 023 Software Cost Estimating
4. DAU CLE 076 Introduction to Agile Software Acquisition
5. ACQ 1700 Agile for DoD Acquisition Team Members
6. "Software is Never Done: Refactoring the Acquisition Code for Competitive Advantage," Defense innovation Board 03 MAY 2019
7. Titanium Card: https://www.dau.edu/tools/titanium
8. https://www.milsuite.mil/book/docs/DOC-954406

TOPIC LEARNING OBJECTIVES

Upon successful completion of this topic, the student will be able to:

12. Identify key discriminators for selecting the most capable software developer.
13. Identify developer practices essential for creation of high-quality software.
14. Using DoD Practical Software Measurement methodology principles, select appropriate software measures to make sound decisions regarding acquisition of software-intensive systems.
15. Recognize three categories of software metrics.
16. Recognize the fiscal impact of Post Deployment Software Support (PDSS).

STUDENT PREPARATION

Student Support Material

1. None

Primary References

1. Defense Acquisition Guidebook, Chapter 3
2. MIL-STD-498 Software Development and Documentation

Additional References

1. DAU ISA 1011 Basic Information Systems Acquisition
2. DAU ISA 201 Intermediate Information Systems Acquisition
3. DAU CLB 023 Software Cost Estimating
4. DAU CLE 076 Introduction to Agile Software Acquisition
5. ACQ 1700 Agile for DoD Acquisition Team Members
6. “Software is Never Done: Refactoring the Acquisition Code for Competitive Advantage,” Defense innovation Board 03 MAY 2019
7. Titanium Card: https://www.dau.edu/tools/titanium
8. https://www.milsuite.mil/book/docs/DOC-954406

Overview

- Common issues
- Cost estimation
- Development and testing
- Software development models
- Software developer selection
- Metrics
- Software support

Software-Intensive System Common Issues

- Program Managers report that software problems typically come from the following sources:
  - Poor requirements definition
  - Lack of user involvement
  - Poorly-defined architecture and interfaces
  - Overlooking hardware deficiencies
  - Failure to establish and/or maintain a functional team of vendors, experts, and end users
- Mission Impacts
  - Delays to software delivery can delay ship/boat delivery
  - Delays to upgrades can create stop work due to lack of access of software programs

Are these issues unique to software?

Software-Intensive System Best Practices

- DoD has identified the following key best practices to follow in the acquisition of software-intensive systems:
  - Risk
    - Identify and manage risk continuously throughout the life-cycle
    - Use metrics to monitor risk, identify problems, and base decisions
    - Address the risks of reusing existing software, whether commercial or non-development items
  - Cost
    - Estimate cost and schedule empirically
    - Track earned value
  - Configuration Control
    - Implement an effective configuration management process
    - Inspect requirements and design; subject configuration management products to formal inspection

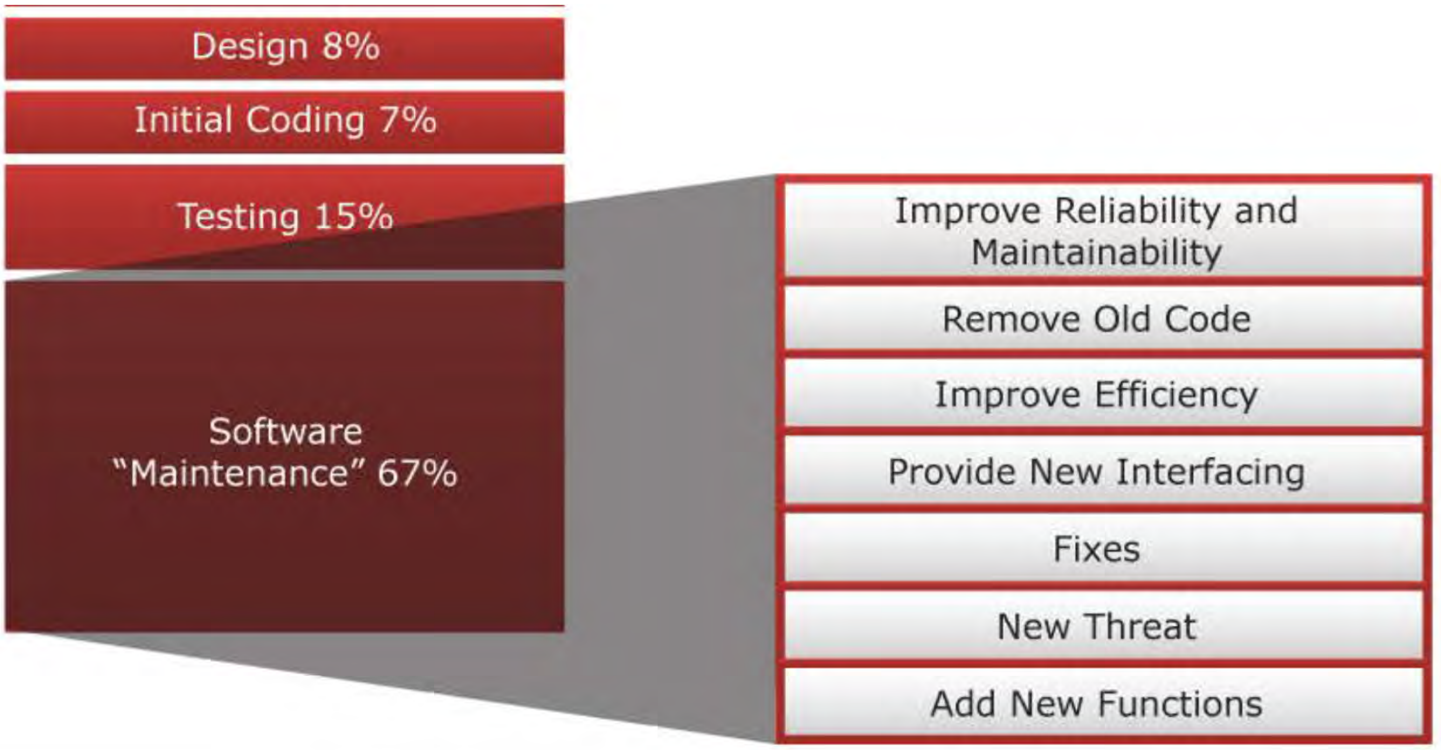
Build the right thing vs building what is needed via timely and frequent user feedback

Software-Intensive System Best Practices

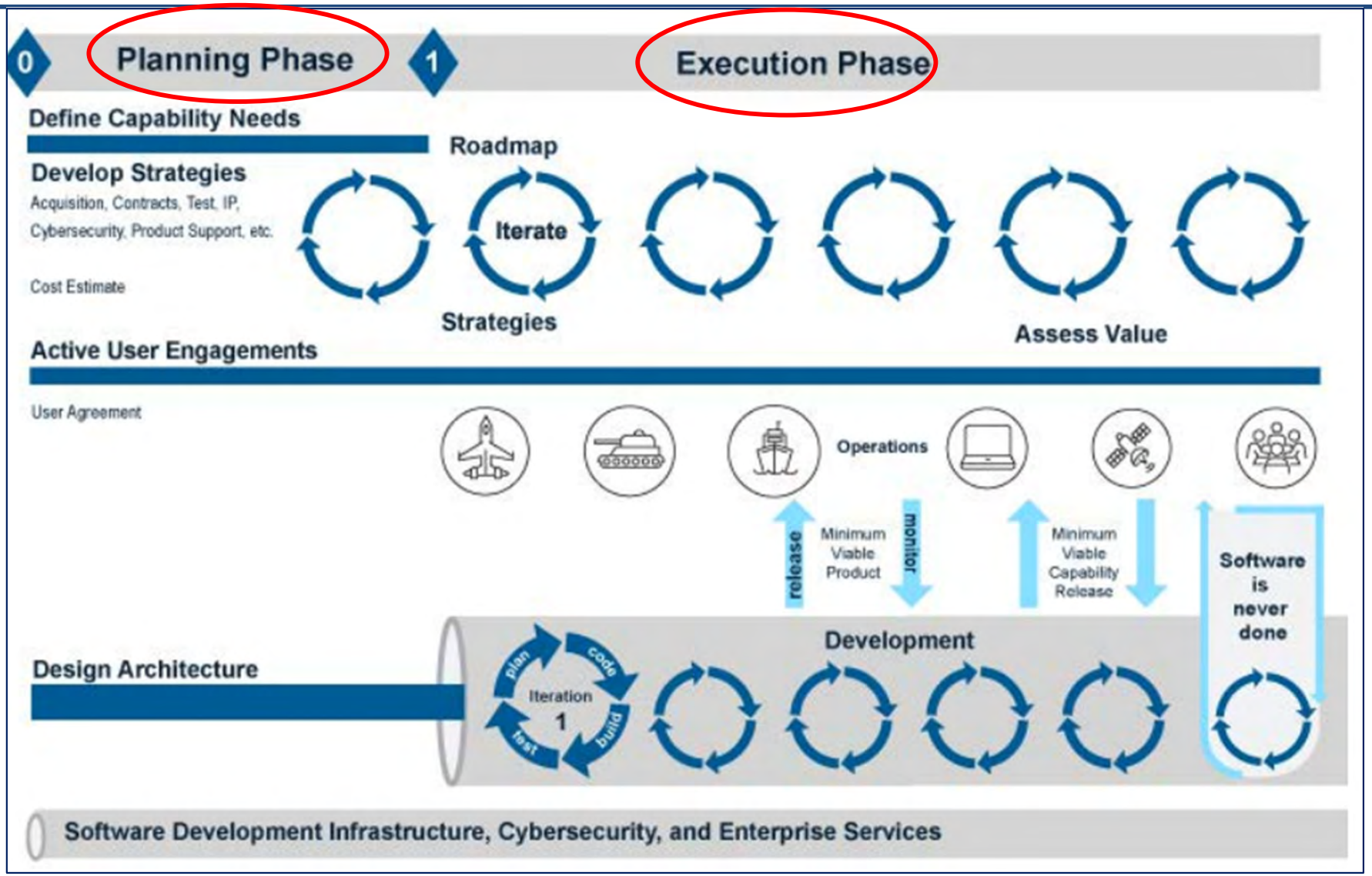
- Life-Cycle Cost
  - Plan early; identify anticipated areas of software changes and enhancements
  - Address cost, performance, and schedule
  - Establish software support concepts and acquire necessary Post Delivery Software Support (PDSS) resources
  - Address software maintenance concepts for systems that include Commercial Off-The-Shelf (COTS) products
  - Include PDSS in planning documentation
- Miscellaneous
  - Manage and trace requirements to the lowest level
  - Ensure data and database interoperability
  - Conduct continuous testing based on plans, pass/fail criteria and traceable procedures
  - End of Service/End of Life considerations and License cost considerations
  - Treat people as your most important resource

Software Cost Breakdown

- Like hardware, the largest distribution of software cost will be late in the program's life-cycle, with decisions made early in the program affecting those final costs



Phases of SW Acquisition



Planning Phase

- Focuses on understanding the users' and systems' needs and planning the approach to deliver capacities to meet those needs
- Information required to enter the Planning Phase:
  - Acquisition Decision Memorandum (ADM)
    - Documents key decisions, program direction, and action items
  - Draft Capability Needs Statement
    - A high-level capture of mission deficiencies, or enhancements, to existing operational capabilities, features, interoperability needs, legacy interfaces, and other attributes that provides enough information to define various software solutions as they relate to the overall threat environment

*The Planning Phase is guided by the **Capability Needs Statement***

Planning Phase

- Artifacts required prior to entry into Execution phase:
  - Capability Needs Statement
  - User Agreement
    - A commitment between the sponsor and PM for continuous user involvement and assigned decision making authority in the development and delivery of software capability releases
  - Cost Estimates and Cost Analysis Requirements Description (CARD)
  - Acquisition Strategy
  - Test Strategy
  - Cybersecurity Strategy
  - Contracting Strategy
  - Intellectual Property (IP) Strategy
  - Program Support Strategy

Acquisition Strategy Planning

- The Program Manager must address the following key life-cycle planning issues for software procurement:
  - Data rights
  - Security
  - System integration
  - Open systems architecture
  - Software re-use
  - Logistics support
  - Software support (includes licensing)
  - Software modification (for emerging capabilities and missions)

Intellectual Property Strategy *and* ***Information Support Plan*** *address these issues*

Contracting/Solicitation Planning

- Software quality requirement
  - The Program Office will work with end-user to generate the requirements
- Software life-cycle development and support environment requirement
  - An automated, computer-based software life-cycle development and support environment will be used by the Contractor
- Data Rights
  - Determination must be made as to amount of data rights to be purchased
- Contracting considerations
  - Should support small, frequent releases, and responds to change (i.e., modular contracting)

Overview

- Common issues
- Cost estimation
- Development and testing
- Software development models
- Software developer selection
- Metrics
- Software support

Cost Estimating

- Cost estimates:
  - Must be developed before the execution phase begins and must be updated annually
  - Tailored for unique aspects of software development and should be
    - Timely
    - Iterative
    - Informative
    - Flexible
    - Credible

Cost Estimating

- Analog
  - Draws a direct comparison between the program being estimated and a similar, completed program
- Parametric
  - Applies regression analysis to a database of several similar systems to develop cost estimating relationships (CERs)
- Engineering Build-up
  - Derived by summing detailed cost estimates of the individual work packages and adding appropriate burdens
  - Most commonly used for software estimates if data exists
- Actual
  - Extrapolates future estimated costs from actual costs; similar to analogy but based on data from the same program

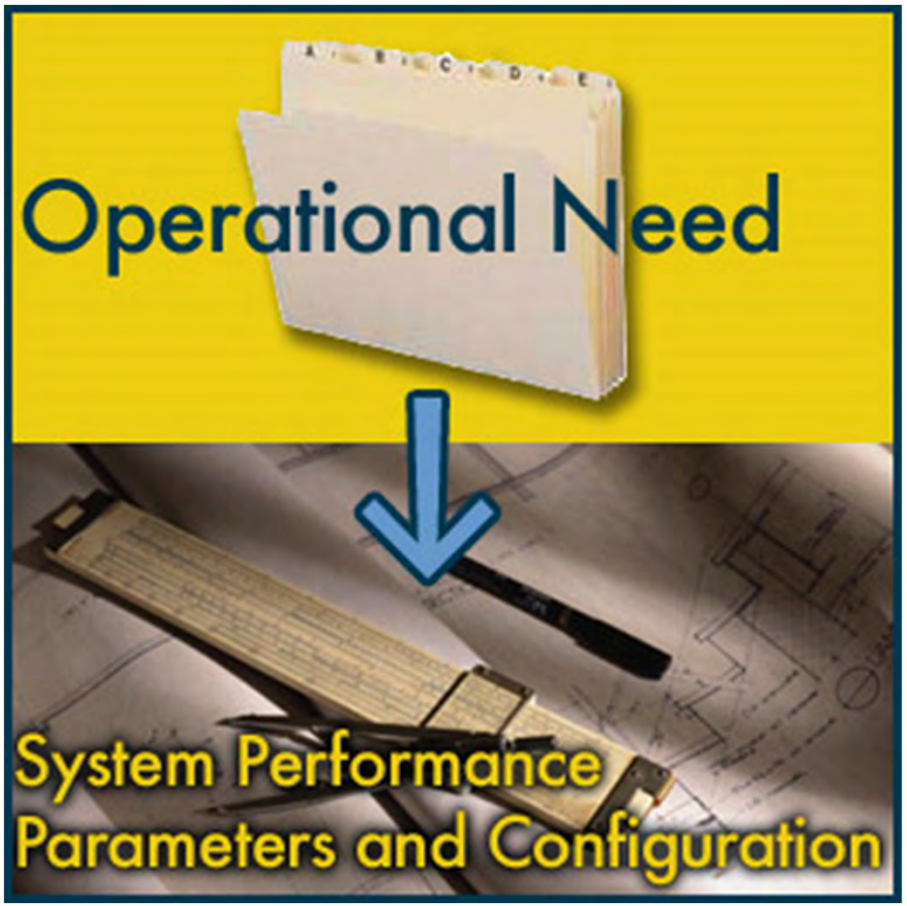
Overview

- Common issues
- Cost estimation
- Development and testing
- Software development models
- Software developer selection
- Metrics
- Software support

Execution Phase

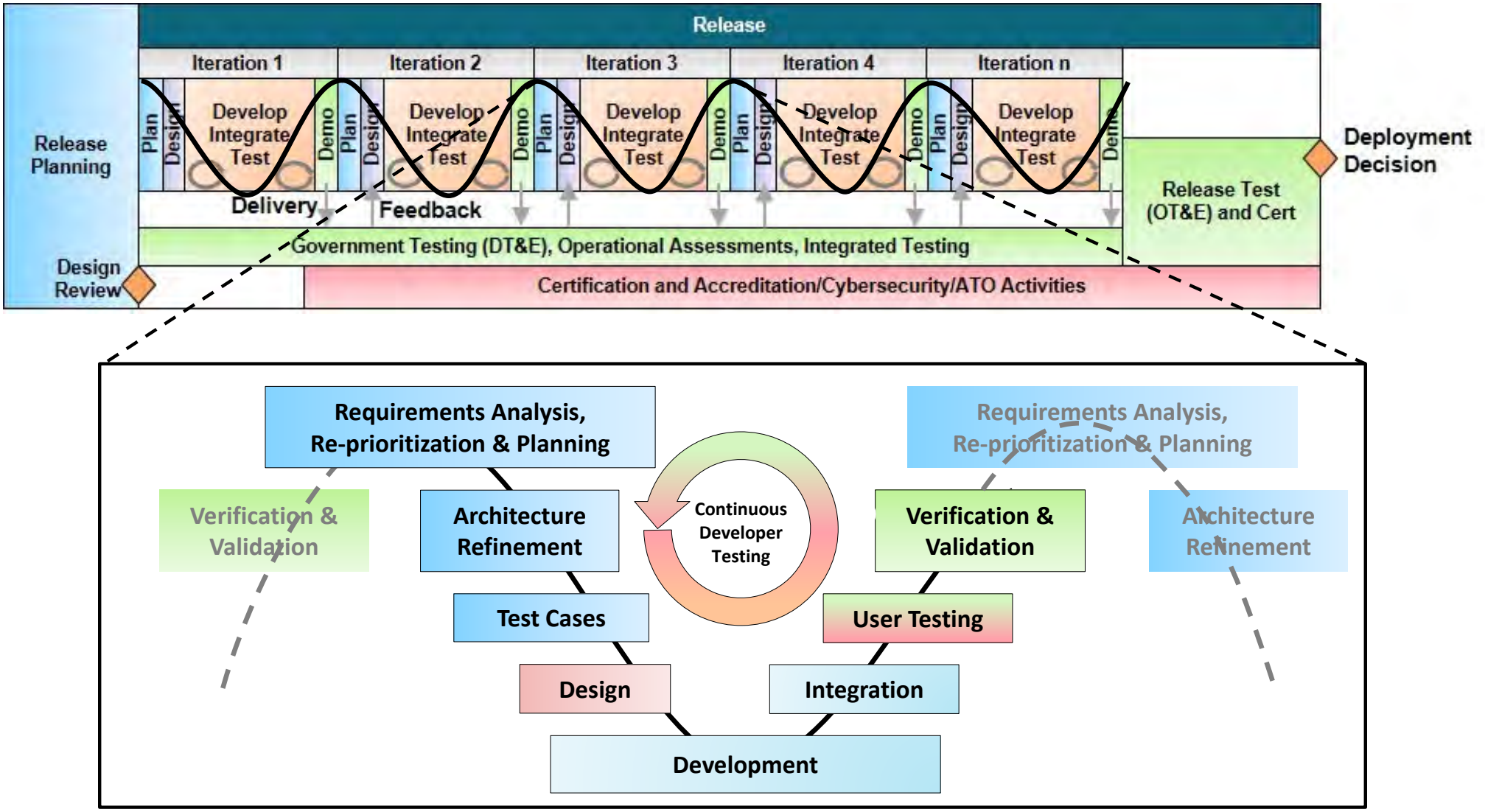
- The purpose of the Execution Phase is to rapidly and iteratively design, develop, integrate, test, deliver, and operate resilient and reliable software capabilities that meet the users' priority needs
- Execution Phase artifacts:
  - Periodic updates (if applicable) to all Planning Phase artifacts
  - Systems architecture
  - Product roadmap
    - Goals and features of each SW iteration and increment
  - Program backlog
    - Prioritized lists of user needs
  - Values assessment
  - Clinger Cohen Act (CCA) compliance
  - Test plans
  - Program metrics

Systems Engineering Process



- Transforming operational needs into a definition of system performance parameters and a preferred system-level configuration
- DoD policies require a program manager to base software systems design and development on systems engineering principles

Software System Engineering "V"



Software Development Activities

- System Requirements Analysis & System Design
  - Develops complete and consistent top-level specifications
  - Determine which requirements are best performed through software
- Requirements Analysis
  - Finalize the selection of Software Items (SI)
  - Determine detailed requirements for each SI*
- Software Design
  - Detailed designs are built for each SI
  - Verify all requirements are met*
- Coding and Implementation
  - Developer writes code for each software unit
    - Each SI may have multiple software units
  - Each software unit is tested in a stand-alone mode

*Not applicable when using Agile development

Software Development Activities

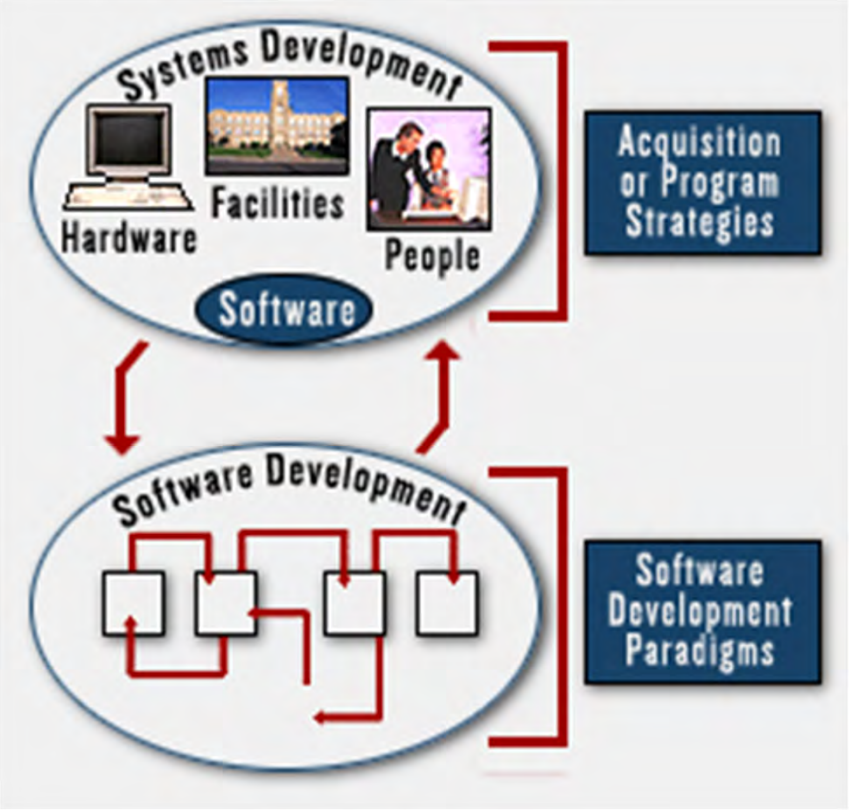
- Unit Integration and Testing
  - Involves progressive integration of software units making up an SI
  - Testing software units with one another, to include internal interfaces
- SI Qualification Testing
  - Test of SI as an entity
  - Described and detailed in the Software Requirements Specification (SRS)
- SI Integration and Testing
  - Integration of various SIs, or even Hardware Items (HIs), that make up a subsystem
- Subsystem Qualification Testing
  - Integration, testing, and qualification of various subsystems

Overview

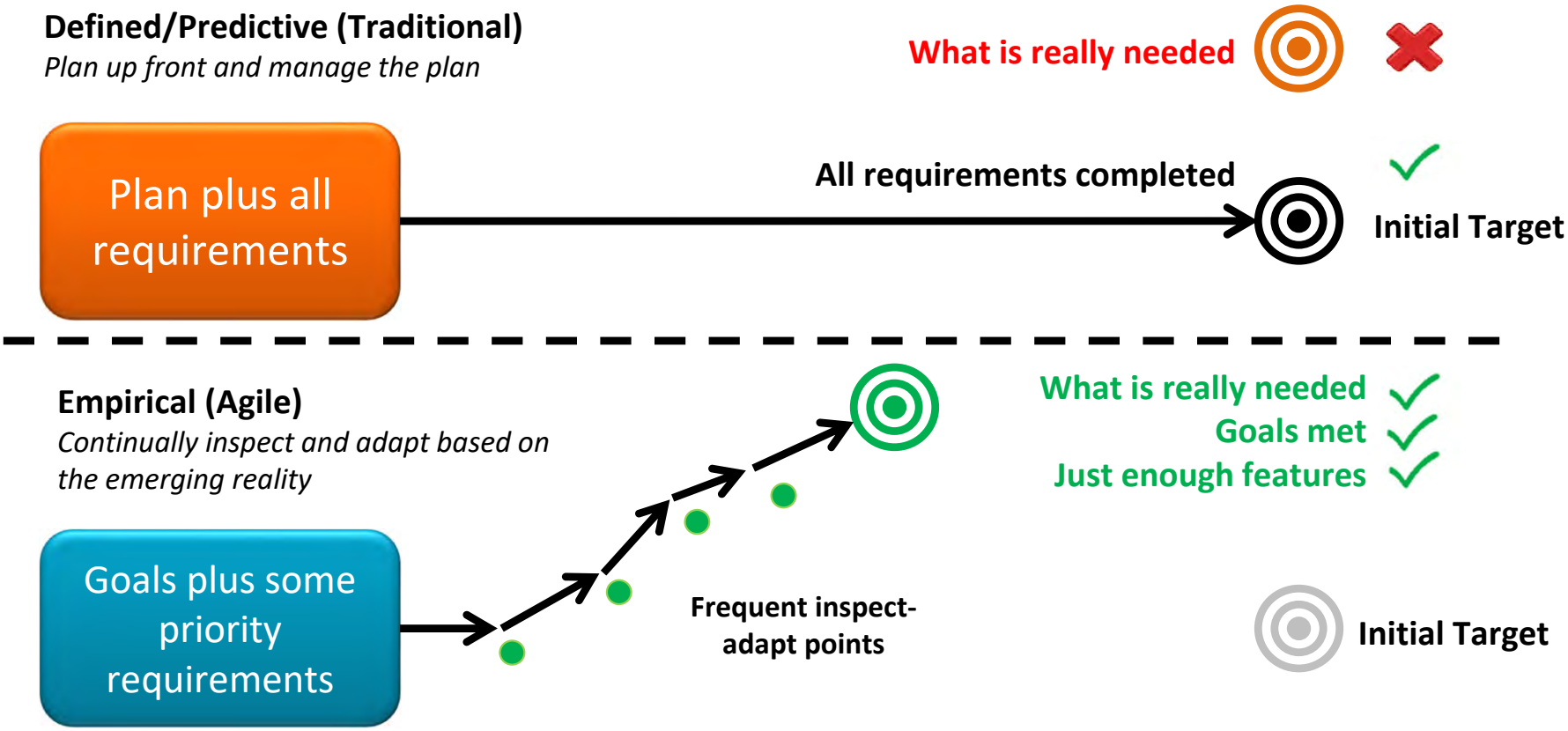
- Common issues
- Cost estimation
- Development and testing
- Software development models
- Software developer selection
- Metrics
- Software support

Software Development Models

- The model selected depends on the acquisition strategy and the type of system being developed
- DoD prefers ***Empirical (Agile)*** development over "traditional" models as it enables continuous inspection and adapting based on emerging threats



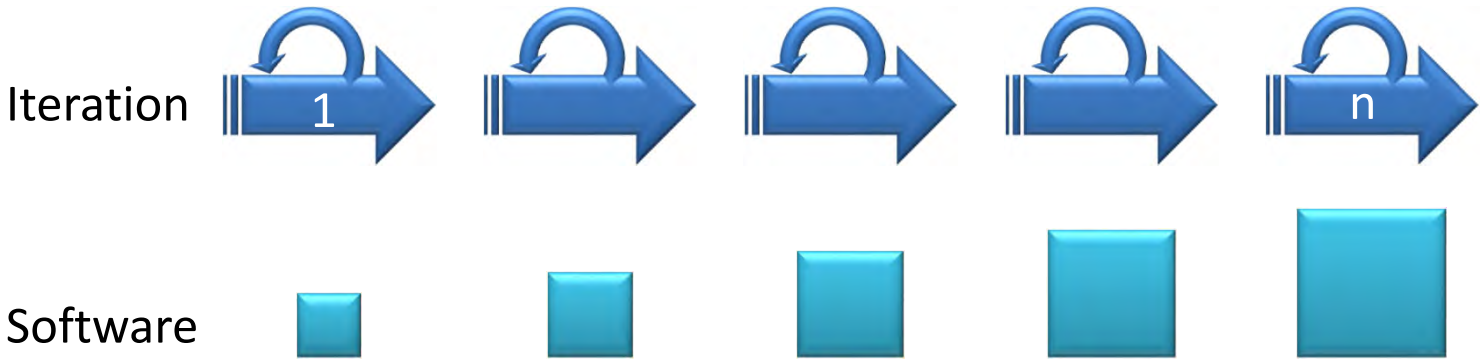
Agile vs Traditional



"Simply delivering what was initially required on cost and schedule can lead to failure in achieving our evolving national security mission — the reason defense acquisition exists in the first place." ~Honorable Frank Kendall, former USD AT&L

Agile Development Definition

A set of software development methods in which requirements and solutions evolve through ***collaboration between self-organizing, cross-functional teams, and end-users***. It promotes adaptive planning, evolutionary development, ***early delivery***, and continuous improvement, and it encourages ***rapid and flexible response to change***



We must be responsive to change

Overview

- Common issues
- Cost estimation
- Development and testing
- Software development models
- Software developer selection
- Metrics
- Software support

Software Process vs Software Process Maturity

- Software Process
  - A set of activities, methods, and practices that people use to develop and maintain associated software products (e.g., plans, requirements, design documents, source code, test cases, manuals)

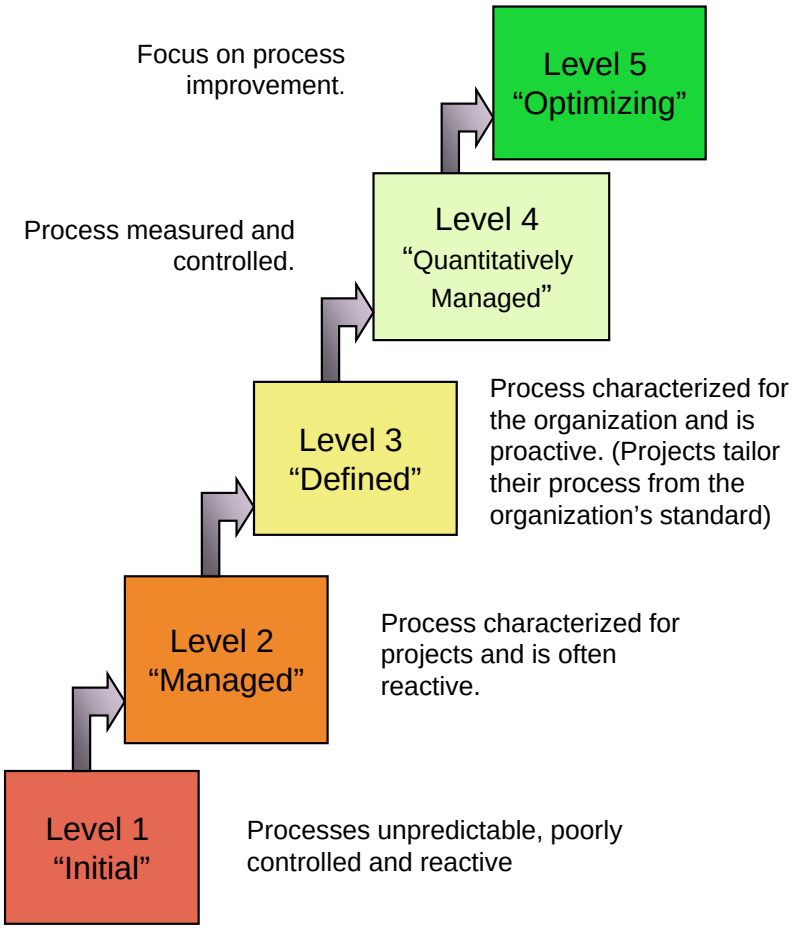
- Software Process Maturity
  - The extent to which a specific process is explicitly documented, managed, measured, controlled, and continually improved

Selecting a Software Developer

- Key discriminators:
  - Organization & resource management
  - Software and systems engineering process management
  - Tools and techniques
  - Software development expertise
  - Experience in the domain or product line
  - Capability Maturity Model – Integrated (CMMI) accreditation
    - Even if the Contractor is not CMMI-rated, a program office can evaluate their capability using a Standard CMMI Appraisal Method for Process Improvement (SCAMPI)
- Good common practices
  - Use of a software measurement process to plan and track software development
  - Use of a mature development process
  - Development of system architectures that support open system concepts
  - Exploitation of existing commercial products
  - Training in software development tools and environments

CMMI Model

Maturity Levels Characteristics



Capability Maturity Model – Integrated

Level	Focus	Process Areas	Result
5 Optimizing	***Continuous process improvement***	Organizational Innovation & Deployment Causal Analysis and Resolution	**Productivity & Quality**
4 Quantitatively Managed	***Quantitative management***	Organizational Process Performance Quantitative Project Management	
3 Defined	***Process standardization***	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Management Risk Management Decision Analysis and Resolution	
2 Managed	***Basic project management***	Requirements Management Project Planning Project Monitoring & Control Supplier Agreement Management Measurement and Analysis Process & Product Quality Assurance Configuration Management	
1 Initial	***Competent people and heroics***		

Software processes in ACAT-I programs should be rated at least Maturity Level 3

Defense Innovation Board
Ten Commandments of Software

1. Make computing, storage, and bandwidth abundant to DoD developers and users
2. All software procurement programs should start small, be iterative, and build on success—or be terminated quickly
3. The acquisition process for software must support the full, iterative life cycle of software
4. Adopt a DevSecOps culture for software systems
5. Automate testing of software to enable critical updates to be deployed in days to weeks, not months or years
6. Every purpose-built DoD software systems should include source code as a deliverable
7. Every DoD system that includes software should have a local team of DoD software experts who are capable of modifying or extending the software through source code or API access
8. Only run operating systems that are receiving (and utilizing) regular security updates for newly discovered security vulnerabilities
9. Security should be a first-order consideration in design and deployment of software, and data should always be encrypted unless it is part of an active computation
10. All data generated by DoD systems—in development and deployment—should be stored, mined, and made available for machine learning

Overview

- Common issues
- Cost estimation
- Development and testing
- Software development models
- Software developer selection
- Metrics
- Software support

Metrics Selection Considerations

- What are the risks we are tracking?
- Which measures are required?
- What data to collect? How is the data defined? How much will it cost to get it? What is the frequency of data collection?
- What is the system and software architecture?
- What are the cost and benefits?
- Are the metrics
  - Understandable?
  - Economical?
  - Field tested?
  - Highly leveraged?
  - Useful at multiple levels?
  - Timely?

Best practices for metrics

- Define a minimal set
- Tailor to management needs
- Collect across the life-cycle
- Use metrics to:
  - Identify risks
  - Improve process
  - Evaluate SW practices
- Use automated collection

Metrics should be tailored to the program's risk areas

Software Measurement Metrics

- Process
  - Maturity and robustness of organizational processes that are used to develop software
- Quality
  - Software product attributes that can impact performance, user satisfaction, supportability, and ease of change
- Management
  - Compare actual progress against plans. Can suggest trends, detect trouble early, or trigger the need to make adjustments to plans

Overview

- Common issues
- Cost estimation
- Development and testing
- Software development models
- Software developer selection
- Metrics
- Software support

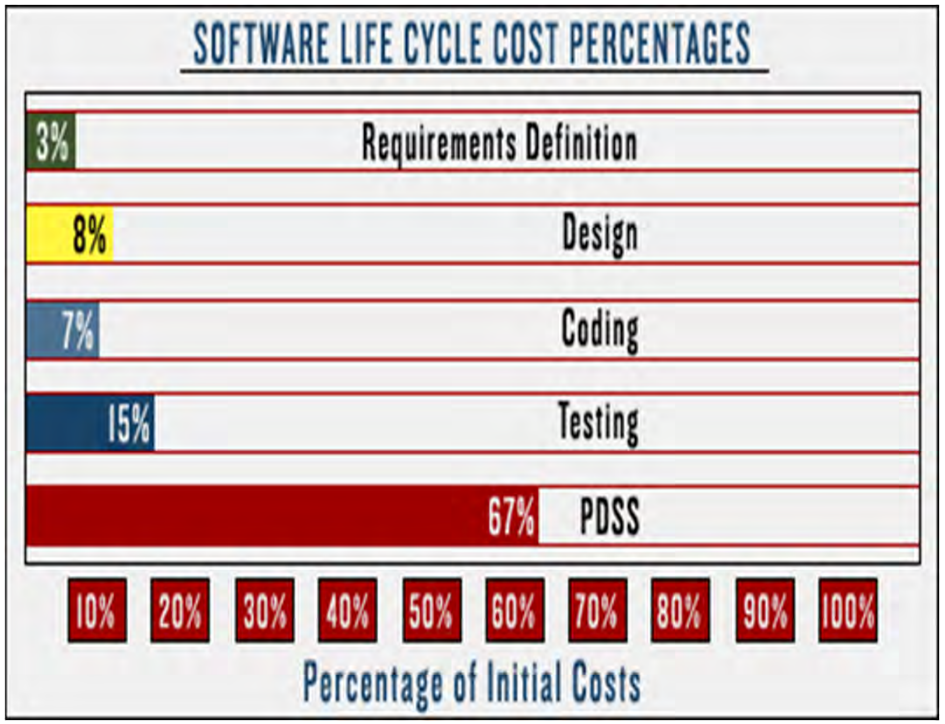
What is Software Support?

- Software Support: The set of activities to ensure that software installed for operational use ***continues to perform as intended*** and fulfill its intended role in system operation. Includes all processes, resources, and infrastructure required to support the operation of software, and to enable its design to be modified in response to new or changed requirements

- Software Maintenance: Modification of a software product ***after delivery to correct faults, to improve performance, or to adapt the product to a modified environment***

Post-Deployment Software Support (PDSS)

- Activities that dominate software support in DoD systems:
  - Improving performance or other attributes
  - Adapting the software to new hardware
  - Adding features and functions to the software to respond to new user requirements
  - Improving efficiency and reliability
  - Correcting errors



PDSS is approximately 67% of the software life-cycle cost

PDSS Best Practices

- Successful Program Managers have learned the importance of early planning for support and maintenance activities. The following lessons have been captured as best practices:
  - Plan early - identify anticipated areas of software changes and enhancements
  - Account for cost, schedule, and performance
  - Establish software support concepts and acquire necessary resources
  - Address software maintenance concepts for systems that include COTS products
  - Include in planning documentation

Summary

- The _____ cost estimating approach is used when a detailed analysis of all materials is required

- What are the 3 categories of software metrics?
  - _
  - _
  - _

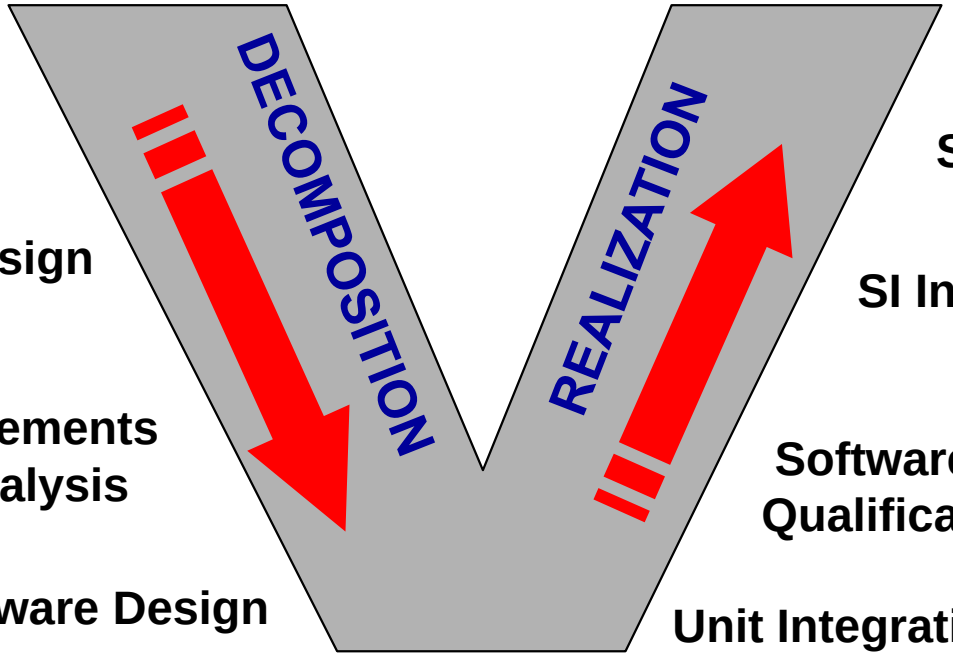
- What is the DoD preferred Software Development Model?
  - _

- Post Deployment Software Support is approximately ___% of total life-cycle cost.

Software System Engineering “V”

Decomposition – “top-down” strategy

Realization – “bottom-up” strategy



System Requirements Analysis

System Design

Software Requirements Analysis

Software Design

Coding & Implementation

Unit Integration & Testing

Software Item (SI) Qualification Testing

SI Integration & Testing

Sub System Qualification Testing