Software Systems Engineering and Rapid Development Methods

GSAW Feb. 29-March 3, 2016
Renaissance Los Angeles Airport Hotel, Los Angeles, CA

Presenters:
David B. Mayo, PhD
Tiara C. Johnson

Space Architecture Department
SAD/ADS/SED

Outline

*The following discussion will highlight maturing software development cycles, project **lifecycles**, and how the government can **adapt** to the ever changing community.*

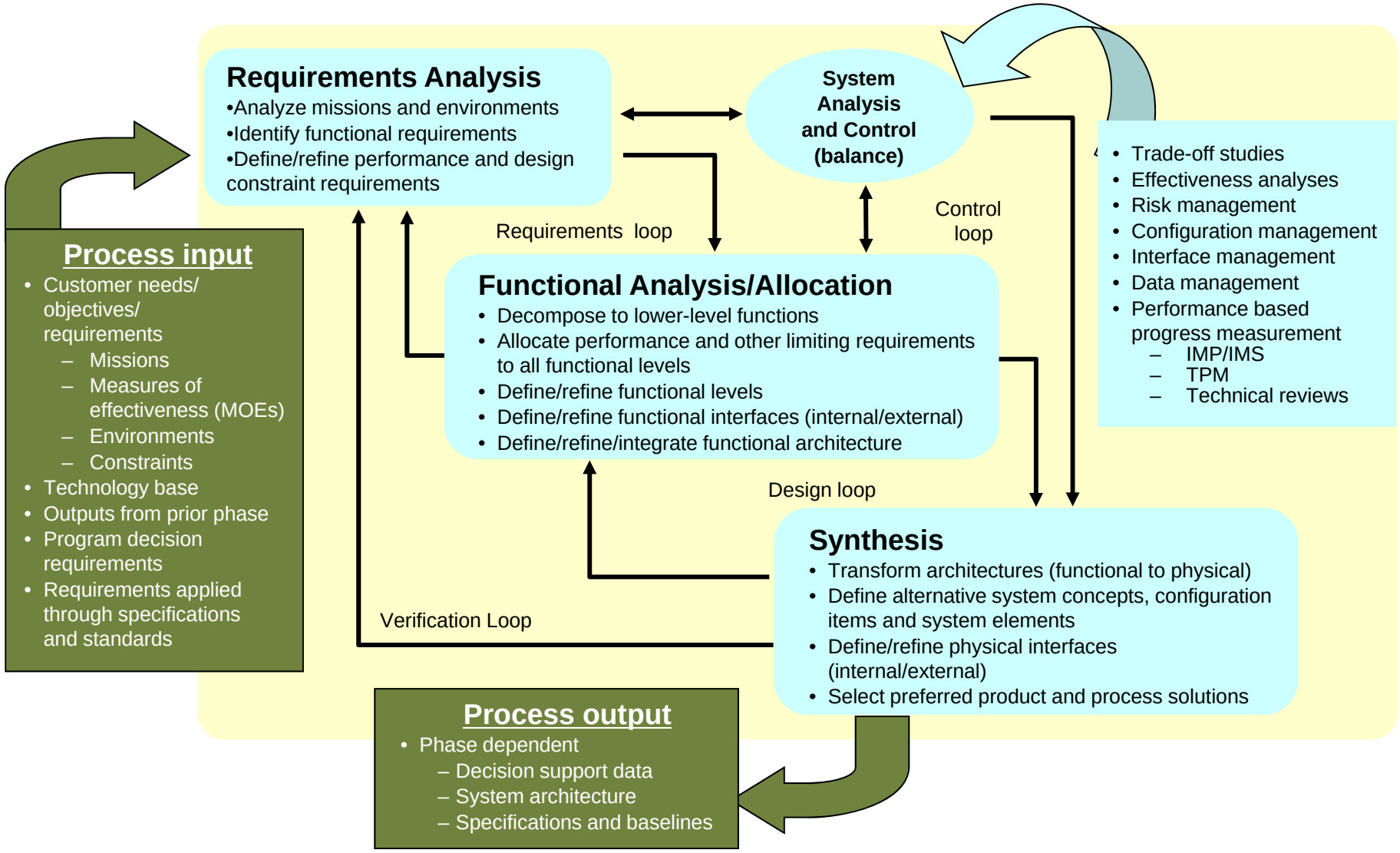
- Introduction/background
  - *Problem Statement*
  - *Summary of Key Topics*
- Recommendations
- Proposal for new readiness review cycles
- Overview of Project Metric Comparison

Software Develop Lifecycles and Systems Engineering

- Traditional systems engineering processes make it difficult to meet the needs of the software development community. This is our motivation for this study.
  - *Faster processes for developing requirements are needed; there is a mismatch in timing between the space vehicle development process and the ground system software development process*

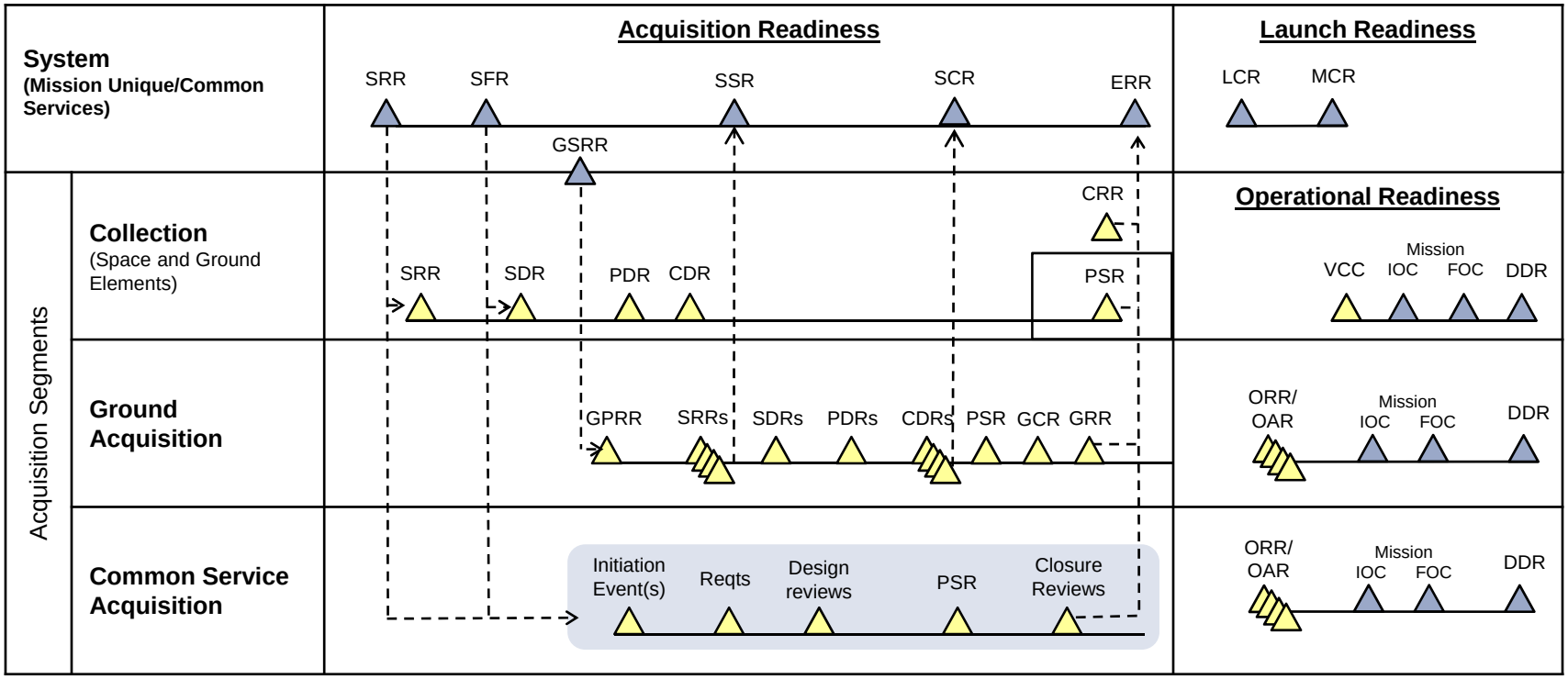
*What **lifecycle models** are out there, and how do we choose the **correct** model for our project?*

Systems Engineering Processes



The basic Systems Engineering processes that need to take place regardless of the software development model or methods

Current Waterfall Development Readiness Reviews



SRR – System requirements Review SFR – System Functional review GSSR – Ground System Readiness Review SSR – System Status Review SCR – System Closure Review ERR – Enterprise Readiness Review LCR – Launch Certification Review	MCR – Mission Certification Review CRR – COMM Readiness Review PSR – Pre-Ship Review GPRR – Ground Project Readiness Review VCC – Vehicle Checkout Complete IOC – Initial Operational Capability FOC – Full Operational Capability	DDR – Deactivation and Disposal Review Reference: Adapted Gov't Lifecycle Readiness Instruction

Incremental and Iterative Software Development

Key Principles

- Incremental and iterative development is a process that grows a system feature by feature during self-contained cycles of analysis, design, development and testing that end in the production of a stable, fully integrated and tested, partially complete system that incorporates all of the features of all previous iterations.

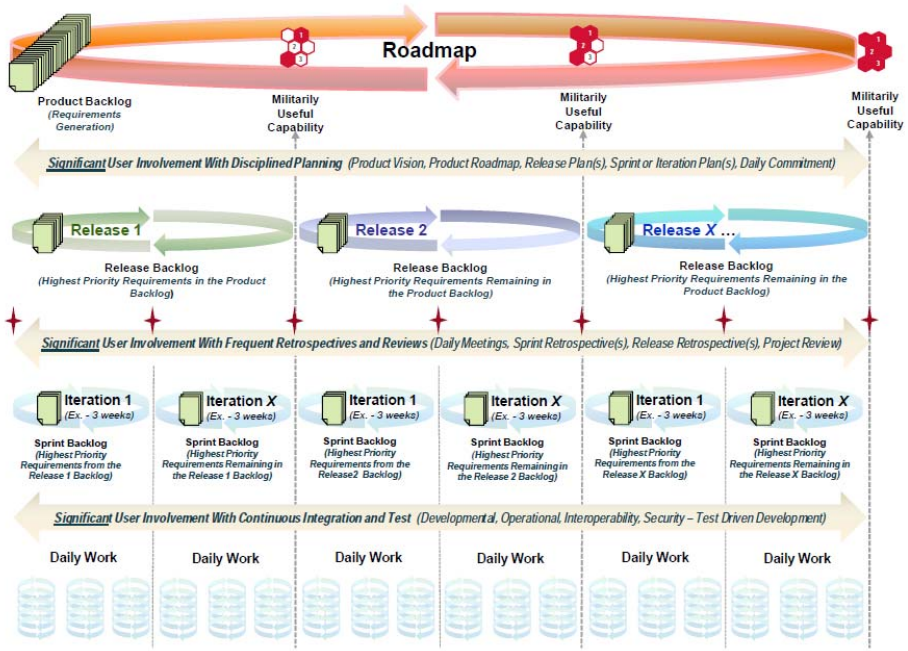
Examples

- Incremental Build Model
- Spiral model
- Agile Software Development
  - *SCRUM*
  - *Extreme Programming (XP)*
  - *Dynamic Systems Development Method (DSDM)*
  - *Crystal*
  - *Feature Driven Development (FDD)*

Examples

- Rational Unified Process (RUP)
- Concurrent Engineering Model
- Rapid Application Development (RAD)
- Joint Application Development (JAD)
- Adaptive Software Development
- Lean Software Development
  - *Kanban*

Agile Software Development



Key Principles

- Customer satisfaction by rapid delivery of useful software
- Regular adaptation to changing circumstances
- Close, daily cooperation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Face-to-face conversation is the best form of communication (co-location)
- Working software is the principal measure of progress
- Self-organizing teams

Benefits

- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Regular adaptation to changing circumstances

Challenges

- Current review system is not compatible with Agile
- Agile uses less documentation
- Government and contractors unfamiliar with Agile
- Culture heavily invested in traditional method
- Progress and value can't be tracked in same way
- Agile requires collaboration and contracting office is not collocated
- Policy to estimate cost based on well known requirements that don't exist in Agile

Potential Challenges in Adopting Non-Traditional Software Development Methods with Government Acquisitions

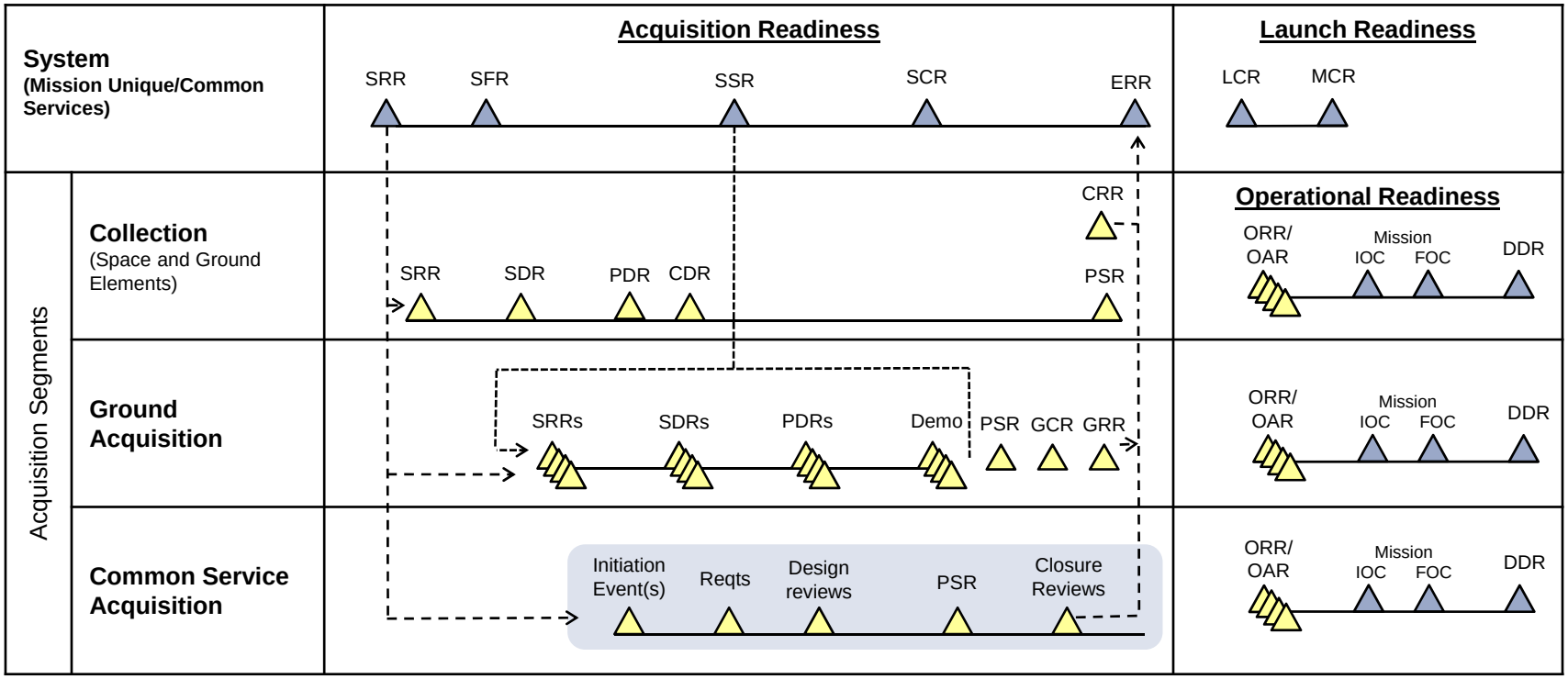
Problem	Potential Solution
Current review process is not compatible with non-traditional methods	- Split milestone reviews into smaller Interim Design Reviews - Modify entry and exit criteria to accommodate artifact maturity
Less documentation produced with non-traditional methods	- Ensure that acquirer understands the documentation process - Negotiate appropriate level of detail for all artifacts
Progress and value can't be tracked in same way with non-traditional methods	- Develop the Integrated Master Schedule (IMS) to an appropriate level of granularity (suggest that an iteration be the lowest level of granularity) - Negotiate with contractors and customer to define suitable progress metrics
Increased team collaboration required for non-traditional methods	- Locate contracting officer on site full time - Rotate (~2 weeks) small teams of customer representatives to the contractor site - Ensure that true users participate in the development process - Identify a single user voice, that can commit to changes for the product being developed, to participate in the development process
System Testing initiated at completion of a traditional development process	- Test incrementally. Engage at the development iteration level

Recommendations for Aligning Systems Engineering Support with Non-Traditional Software Development Methods

Apply these steps when considering how to respond to proposed software development models for a specific program/project

- **STEP 1**: Evaluate status of enterprise level culture and expertise with respect to the proposed software development method
  - *Consider need for top-down fostering of culture and training of personnel*
- **STEP 2:** Document Program/Project characteristics that determine appropriateness of software development methods
  - *Use attached checklist to summarize findings*
- **STEP 3**: Validate that the proposed development methodology is consistent with project characteristics
  - *Compare priorities of project with software development method strengths and weaknesses*
- **STEP 4**: Align Systems Engineering and Lifecycle Readiness Processes with the selected methodology
  - *Define tailored series of readiness reviews to match project characteristics; see attached examples*

Possible Ground Agile Development Readiness Reviews



SRR – System requirements Review SFR – System Functional review GSSR – Ground System Readiness Review SSR – System Status Review SCR – System Closure Review ERR – Enterprise Readiness Review LCR – Launch Certification Review	MCR – Mission Certification Review CRR – COMM Readiness Review PSR – Pre-Ship Review GPRR – Ground Project Readiness Review VCC – Vehicle Checkout Complete IOC – Initial Operational Capability FOC – Full Operational Capability	DDR – Deactivation and Disposal Review Demo – Showing output of latest iteration Reference: Adapted from Gov't Lifecycle Readiness Instruction

For Agile ground software development use a sub-set of the traditional Reviews and iterate as needed.

Metrics to Evaluate the Benefits of Innovative Software Development Lifecycles

- How can we compare similar or identical software projects that use different lifecycles in order to determine the true efficiency or value of using an innovative approach over a standard one?

- The choice for using waterfall development or a new, innovative approach needs to be based on the overall project goals.

Schedule	Scope	Cost	Quality	Risk/Safety
•Number of Software Releases •Cycle time of software releases (amount of time to release) •Unit of work completion number and rate, measured in value in EVM, or story points in agile •Length of Project	•# unit of work, measured in value in EVM, or story points in agile, or CSCI's •Number of Reqs •# of Expected Changed/Altered/Updated Requirements (measures adaptability) •# of Contract Changes •Lines of Code •# of Test cases developed, executed, passed •# of Documents	•Cost •Budget at Completion	•Problem Reports opened, closed •Problem Report closure rate •Average Problem Report closure time	•Number of Tracked Risks •Number of Resolved Risks •Number of Lessons Learned

Comparing metrics across projects

Consider the questions from Step 2 in the Recommendations

Project with similar content will have similar results:
– Are the requirements well-established, or ill-defined? – Are the requirements fixed, or likely to change as the project progresses? – Is the project small to medium-sized (up to 4 people for 2 years) or larger? – Is the application similar to projects that the developers have experience in, or is it a new area? – Is the software likely to be straightforward or complex (e.g. does it use new hardware)? – Does the software have a small easy user interface or a large complex user interface? – Must all the functionality be delivered at once or can it be delivered as partial products? – Is the product safety critical or not?

Similar project content, Similar lifecycle	Similar project content, Different lifecycle
Direct comparison of metrics in all phases. – Cost – Schedule – Budget	Direct comparison of metrics at project boundaries. – Baseline – Launch – Completion
Different project content, Similar lifecycle	**Different project content, Different lifecycle**
Comparison of normalized metrics (relative to “ideal”) during all phases. – Cost – Schedule – Budget	Comparison of normalized metrics (relative to “ideal”) across projects at project boundaries. – Cost Variance – Schedule Variance – Problem Reports

Summary

- Traditional systems engineering processes are not meeting the needs of the software development community in the context of ground systems
- Methods
  - *Incremental/Iterative Software Development Methods*
  - *Agile*
- Proposal for new readiness review cycles and recommendations
  1. Evaluate status of enterprise level culture and expertise with respect to the proposed software development method
  2. Document Program/Project characteristics that determine appropriateness of software development methods
  3. Validate that the proposed development methodology is consistent with project characteristics
  4. Align Systems Engineering and Lifecycle Readiness Processes with the selected methodology

Back-Up

Incremental Software Development

Key Principles

- User requirements allocated to multiple releases
- Initial release includes core functionality (High priority requirements)
- Completed functionality is operationally ready
- Subsequent releases provide additional functionality
- Each release consists of a requirements, design, implementation and testing phase

Benefits

- Decreased “Time to Market” for core capabilities.
- Decreased cost for initial delivery
- Facilitates more targeted and rigorous testing
- Implementation errors more easily identified because of fewer requirements and capabilities in each release
- Easier to accommodate changes in requirements
- Easier to manage risk (high risk requirements are identified and mitigated by release)
- Customer can provide feedback after each release

Challenges

- Requires good initial design and analysis of the entire system in order to define cohesive releases
- Total cost may exceed the cost of traditional development
- Possible system architecture mismatch as additional functionality is added
- Additional (repetitive) regression testing required

Iterative Software Development

Key Principles

- Initial specification of a subset of the total requirements
- Cyclic process of prototyping, testing, analyzing, and refining the requirements and the solution
- Continuous user feedback solicited and used to modify the design of subsequent iterations

Benefits

- The initial design is available earlier for user evaluation
- Allows for concurrent implementation (Overlapping iterations)
- Implementation errors more easily identified
- Easier to accommodate changes in requirements
- User feedback solicited and incorporated in all phases

Challenges

- Total cost may exceed the cost of traditional development
- Possible system architecture mismatch as additional functionality is added
- Poorly defined iteration exit criteria can cause cost and schedule overruns
- Continuous user feedback may result in scope creep

Incremental and Iterative Software Development

Key Principles

- Incremental and iterative development is a process that grows a system feature by feature during self-contained cycles of analysis, design, development and testing that end in the production of a stable, fully integrated and tested, partially complete system that incorporates all of the features of all previous iterations.

Examples

- Incremental Build Model
- Spiral model
- Agile Software Development
  - *SCRUM*
  - *Extreme Programming (XP)*
  - *Dynamic Systems Development Method (DSDM)*
  - *Crystal*
  - *Feature Driven Development (FDD)*

Examples

- Rational Unified Process (RUP)
- Concurrent Engineering Model
- Rapid Application Development (RAD)
- Joint Application Development (JAD)
- Adaptive Software Development
- Lean Software Development
  - *Kanban*

Free & Open Source Software

Key Principles

- Open Source Software is software distributed under terms maintained by the Open Source Initiative (OSI)
- Human readable source code is available and freely distributed (allowed in both compiled and source form)
- Software is redistributable (subject to licenses, it may be sold)
- Derived works are allowed, but often must be distributed under the same terms as the original license ('Viral' Licensing)
- Licensing provide means for distribution, modification, and use
- Enables community *writ large* access to develop key methods or components

Benefits

- Potential to reduce development times through use of pre-existing tools
- Continuous and broad peer-review supports reliability and security
- Unrestricted ability to modify source code enables adaptability toward changing situation, mission and threats
- Reliance on singular developer or vender due to proprietary restriction may be reduced (OSS maintenance from multiple vendors, reduce barrier to entry)
- OSS licenses do not restrict who or what fields can use the software: rapid provisioning of known and unanticipated users

Challenges

- Licensing can be a complication
- "Free" – No warranties expressed or implied when using the software. Bugs can, and often will, occur, OSS projects mitigate risk of bugs using tools and processes, Companies will often sell tech. support for their OSS
- Focus on working software over comprehensive documentation
- Code itself is often seen as the 'documentation'
- Open means open: Anyone who can access the code or project can potentially contribute
- Usually contributions are vetted only for accuracy (expected input/output)

Model Based Systems Engineering

Key Principles

- MBSE is Model-Centric rather than Document-Centric
- It's not modeling and simulation, or just using models – it's using models as the method of recording your design.
- Traditional Systems Engineering uses documents to describe systems
  - *System requirements, system design, interface requirements, sub system requirements, etc. are all contained in documents*
- MBSE uses models to describe systems
  - *System requirements, system design, interface requirements, sub system requirements, etc. are all contained in model(s)*

Benefits

- Higher productivity
- Easier to verify the design
- Both the system and software design can leverage the modeling tool for design verification
- Increases design quality
- Increased interoperability: abstract higher level model used to generate the detailed lower level models
- Reduced maintenance – no document maintenance, just design maintenance

Challenges

- Inadequate tool support (over 50 tools used, no current market leader, expensive, open source tools not) capable of meeting the needs
- Tool integration difficult
- Government must purchase licenses & training for tools
- Must know how to write RFP and contract when MBSE is used
- Cultural changes are required: CDRLs are models, not documents
- Challenges with autocode, such as the lack of optimization
- Lack of standardized MBSE Metrics

Recommendations for Aligning Systems Engineering Support with Non-Traditional Software Development Methods

STEP 1: Evaluate status of enterprise level culture and expertise with respect to the proposed software development method

- **Knowledge of Agile Principles, Benefits, and Risks**
  - *Challenges*
    - *Lack of Familiarity with Agile Among Acquisition Professionals*
    - *Perception that Agile Equals Higher Risk*
  - *Solutions*
    - *Increase Knowledge through Educational Sessions and a Myth-Busting Campaign*
    - *Expose Acquisition Professionals to Agile Development Products*
    - *Develop Agile Procurement Coaches*
    - *Refocus Attention on "Top 4 Risks"*
- **Stakeholder Ownership & Decision Making**
  - *Challenges*
    - *Lack of Empowerment and Accountability*
    - *Lack of Commitment and Engagement*
  - *Solutions*
    - *Identify and Empower Stakeholders Early*
    - *Product Owner as a Near Full-time Role*
    - *Product Owner as Career Building Role*

Recommendations for Aligning Systems Engineering Support with Non-Traditional Software Development Methods

STEP 2: Document Program/Project characteristics that determine appropriateness of software development methods by answering questions such as those given below

Sample Questions

- Are the requirements well-established, or ill-defined?
- Are the requirements fixed, or likely to change as the project progresses?
- Is the project small to medium-sized (up to 4 people for 2 years) or large?
- Is the application similar to projects that the developers have experience in, or is it a new area?
- Is the software likely to be is it straightforward or complex (e.g. does it use new hardware)?
- Does the software have a small easy user interface or a large complex user interface?
- Must all the functionality be delivered at once or can it be delivered as partial products?
- Is the product safety critical or not?
- Are the developers largely inexperienced or mainly experienced?
- Does the organizational culture promote individual creativity and responsibility or does it rely on clear roles and procedures?

Recommendations for Aligning Systems Engineering Support with Non-Traditional Software Development Methods

STEP 3: Validate that the proposed development methodology is consistent with project characteristics

- *Compare priorities of project with software development method strengths and weaknesses*
- *Example descriptions of software development methods with their strengths and weaknesses are given in the following 2 slides*

Step 3: Lifecycle Model Definitions & Applications

Waterfall

Development Method	Most Appropriate	Least Appropriate
Waterfall Traditional method of project lifecycle. Phases include: Initiation, Planning, Execution, Monitoring and Controlling and Closing. Requirements are documented in detail, up front, followed by refinement in the system and then testing – in a "waterfall" fashion.	- Project is for development of a mainframe-based or transaction-oriented batch system. - Project is large, expensive, and complicated. - Project has clear objectives and solution. - Pressure does not exist for immediate implementation. - Project requirements can be stated unambiguously and comprehensively. - Project requirements are stable or unchanging during the system development life cycle. - User community is fully knowledgeable in the business and application. - Team members may be inexperienced. - Team composition is unstable and expected to fluctuate. - Project manager may not be fully experienced. - Resources need to be conserved. - Strict requirement exists for formal approvals at designated milestones.	- Large projects where the requirements are not well understood or are changing for any reasons such as external changes, changing expectations, budget changes or rapidly changing technology. - Web Information Systems (WIS) primarily due to the pressure of implementing a WIS project quickly; the continual evolution of the project requirements; the need for experienced, flexible team members drawn from multiple disciplines; and the inability to make assumptions regarding the users' knowledge level. - Real-time systems. - Event-driven systems. - Leading-edge applications.

Step 3: Lifecycle Model Definitions & Applications

Iterative & Incremental

Development Method	Most Appropriate	Least Appropriate
Incremental and iterative development is a process that grows a system feature by feature during self-contained cycles of analysis, design, development and testing that end in the production of a stable, fully integrated and tested, partially complete system that incorporates all of the features of all previous iterations. Examples Include: • Incremental Build Model • Spiral model • Agile Software Development • Rational Unified Process (RUP) • Concurrent Engineering Model • Rapid Application Development (RAD) • Joint Application Development (JAD) • Adaptive Software Development • Lean Software Development	**Iterative**	
	- Project is for an online system requiring extensive user dialog, or for a Less well-defined expert and decision support system. - Project is large with many users, interrelationships, and functions, where project risk relating to requirements definition needs to be reduced - Project objectives are unclear. Pressure exists for immediate implementation of something. - Functional requirements may change frequently and significantly. - User is not fully knowledgeable. - Team members are experienced (particularly if the prototype is not - Team composition is stable & Project manager is experienced. - No need exists to absolutely minimize resource consumption. - No strict requirement exists for approvals at designated milestones. - Analysts/users understand the business problems involved, before they begin the project. - Innovative, flexible designs that will accommodate future changes are not critical.	- Mainframe based or transaction oriented batch systems. - Web-enabled e-business systems - Project team composition is unstable. - Future scalability of design is critical. - Project objectives are very clear; project risk regarding requirements is very low.
	Incremental	
	- Large projects where requirements are not well understood or are changing due to external changes, changing expectations, budget changes or rapidly changing technology. - Web Information Systems (WIS) and event driven systems - Leading-edge applications.	-Very small projects of short duration - Integration and architectural risks are low. - Highly interactive applications where the data for the project already exists (completely or in part) and the project largely comprises analysis or reporting of the data

Selecting a Development Approach Dept. of Health and Human Services 2008

Recommendations for Aligning Systems Engineering Support with Non-Traditional Software Development Methods

STEP 4: Align Systems Engineering and Lifecycle Readiness Processes with the selected methodology

Category	Challenges	Solutions
Performance Measurement	• Typical Performance Measures Do Not Measure Customer Satisfaction or Value • Lack of Pre-Defined Documented Standard to Define Acceptance Criteria	• Collaborate with Stakeholders, Agency Leadership, and Office of Management Budget • Focus on Core Capabilities of Software Functionality and Iterative Documentation Development for what is really needed for the moment • Adopt Suitable Cost and Schedule Performance Measures • Measure Quality via Customer Satisfaction which can determine value to the mission
Contract Types	• The Drive towards Firm-Fixed Price (FFP) Scope Contracts – current trend for acquisitions	• Use Time & Material, Cost Plus Fixed Fee, FFP Level of Effort, and Labor Hour Contracts • Avoid Firm Fixed Price Scope Contracts which discourages flexibility/changes or uncertainty in requirements
Internal Government Costs	• Difficulty Accounting for All Government Costs due to undefined roles and responsibilities (R&R)	• Identify, Track, and Quantify Internal Government Costs with well defined R&R • Address the Myth of Administrative Burden on Flexible Projects
Testing and IV&V	• Approach to IV&V May Add Unnecessary Cost	• Set Expectation that IV&V Testers Will Be Integrated into the Project Team • Refocus IV&V towards Quality Control and Process Improvement

- Update the readiness review schedule for the specific software development model, as proposed in the following slides

Comparing metrics across projects

Project Content	Lifecycle: Similar	Lifecycle: Different
Similar	**Similar project content, Similar lifecycle**	**Similar project content, Different lifecycle**
Similar	A project in this category will have a direct comparison of all metrics in all phases.	A project in this category will have direct comparison of metrics only at project *boundaries*.
Similar	**Cost** BCWP, ACWP, CV, CPE, EAC; **Schedule** PV, EV, SV; **Scope** Work planned, Work to complete	PV & Budget at **Baseline**; SV, AC, at **Launch**; AC, EV at **Completion**
Different	**Different project content, Similar lifecycle**	**Different project content, Different lifecycle**
Different	A project in this category will need to use a comparison of *normalized* metrics (relative to "ideal") during all phases. Use indices for normalization.	A project in this category will need to use a comparison of *normalized* metrics (relative to "ideal") across projects only at project *boundaries*.
Different	**Cost** CPI; **Schedule** SPI; **Scope** Remaining Milestones/Completed Milestones	Cost Variance at **Baseline**; Schedule Variance at **Launch**; Number of Problem Reports at **Completion**

Apply the metrics as appropriate based on whether the project content and project lifecycle models are similar or different

Healthcare Marketplace Failure
A Case Study in a Failed Agile Approach

Background

- Patient Protection and Affordable Care Act passed in March 2010. Law required operational marketplaces by Jan 1, 2014
- Healthcare.gov was to be the federal marketplace used by US residents to shop for health insurance in states without their own healthcare marketplaces
- First of its kind federal marketplace was a complex effort exacerbated by compressed time frames and changing requirements. Failures included
  - *Significant cost increases*
  - *Schedule slips*
  - *Delayed system functionality*
  - *Inadequate verification prior to release*
- Results
  - *Non functioning marketplaces at time of release*
  - *Extension of enrollment deadline*
  - *Brought in new contractors to fix the product leading to even higher costs*
- Current status
  - *Now runs smoothly for most users*
  - *End of open enrollment*
  - *8 million people signed up for private health insurance in the first year (using state and federal sites)*

Project Challenges

- Key requirements not defined
  - *Requirements for state support unknown*
  - *Requirements for main functionality finalized after contract awarded*
  - *Ongoing regulation and policy changes led to changes in requirements outside project control*
- Compressed timeframe
  - *Only 3.5 years to perform acquisition, and development and testing*
    - Study showed pre-solicitation planning activities required could last more than 2 years
  - *States didn't have to declare their intent until 10 months prior to delivery*
    - Didn't know size of users base

Main points of failure

- Issued task orders before key requirements were defined
- Implemented Agile without preparation
  - *No training or previous experience*
  - *No adapted procurement strategy*
  - *No single customer voice*
  - *No risk analysis*
  - *Inadequate milestone review plan and action*
    - Delayed or skipped some reviews
    - Began testing and validation only 1 month before release
- Cost reimbursed contracts
  - *Created additional risk*
  - *Had to pay even when functionality not delivered*
  - *Increasing fees*
- No action when performance issues arose
  - *Resulted in the release of non verified product*
- Lack of proper oversight
  - *Confusion about who had authority to approve contractor requests to extend funds*
  - *Limited steps to hold contractor accountable*
  - *Incomplete acquisition strategy*
  - *Required quality assurance plan not used*

Application of Software SE study to case study

Lessons Learned

- They leapt into Agile without proper preparations
  - *Must train all involved in Agile process*
  - *Must evaluate if Agile if right for the project*
  - *Must alter acquisition process to fit Agile*
    - Includes new risk assessment
  - *Must alter milestone reviews to fit project*
  - *Must have a single customer voice*
  - *"Flexible requirements" does not mean undefined requirements*

Sources & References

- “Acquisition best practices to procure IT services.” Emerging Technology Shared Interest Group. 2014.
- Dove, Rick. "Fundamental principles for agile systems engineering." *Conference on Systems Engineering Research (CSER), Stevens Institute of Technology, Hoboken, NJ*. 2005.
- Haberfellner, Reinhard, and Olivier de Weck. "Agile systems engineering versus agile systems engineering." *INCOSE 2005 Symposium*. 2005.
- Huang, Philip M., Ann G. Darrin, and Andrew A. Knuth. "Agile hardware and software system engineering for innovation." *Aerospace Conference, 2012 IEEE*. IEEE, 2012.
- Lapham, M. A., et al. *Agile Methods: Selected DoD Management and Acquisition Concerns, October 2011, Technical Note*. CMU/SEI-2011-TN-002.
- Lapham, Mary A., et al. *Considerations for Using Agile in DoD Acquisition*. No. CMU/SEI-2010-TN-002. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2010.
- Modigliani, Pete, and Su Chang*. Defense agile acquisition guide: Tailoring DoD acqusition program structures and processes to rapidly deliver capabilities.* MITRE Corporation. 2014.
- Palmquist, Steve, et al. "Parallel Worlds: Agile and Waterfall Differences and Similarities." (2013).
- “Software Development: Effective Practices and Federal Challenges in Applying Agile Methods.” United States Government Accountability Office. GAO-12-681. 2012
- Russo et al. “Agile Technologies in Open Source Development.” 2009. IGI Global. ISBN-10: 1-59904-681-4. Available via the Aerospace Library (TAL) and Safari Books Online (via TAL’s subscription)
- US DoD CIO. “DoD Open Source Software (OSS) FAQ.” Available http://dodcio.defense.gov/OpenSourceSoftwareFAQ.aspx
- US DoD CIO. “Clarifying Guidance Regarding Open Source Software” and appendices. 2009. Available: http://dodcio.defense.gov/Portals/0/Documents/OSSFAQ/2009OSS.pdf

Sources & References

- Warsta, J., & Abrahamsson, P. (2003). Is open source software development essentially and agile method? 3rd Workshop on Open Source Software Engineering, Portland, OR.
- Scacchi, W. (2002) *Open source software development processes. Version 2.5.* Retrieved on 14 June 2014 from http://www.ics.uci.edu/~wscacchi/Software-Process/Open-Software-Process-Models/Open-Source-Software-Development-Processes.ppt
- Defense Contract Management Agency (DCMA). (2012). *Earned Value Management System (EVMS) Program Analysis Pamphlet (PAP).* Washington D.C.: Department of Defense.
- Koontz, D. (2014, February). *Metrics for a Scrum Team*. Retrieved September 2014, from Scrum Alliance Agile Atlas: http://agileatlas.org/articles/item/metrics-for-a-scrum-team
- NASA. (2009). *NPR 7150.2A NASA Software Engineering Requirements.* Washington D.C.: NASA.
- NASA. (2010). *NPR 7120.5, NASA Space Flight Program and Project Management Handbook.* Washington, D.C. NASA.
- Project Management Institute. (2014). *Earned Value Management*. Retrieved August 2014, from Project Management Institute: http://www.pmi.org/Knowledge-Center/Knowledge-Shelf/Earned-Value-Management.aspx
- Software Engineering and Software Advanced Research Lab (SESAR). (2007). *Metrics for CMMI Maturity Level*. Retrieved September 2014, from CMMI Metrics Framework: http://sesar.di.unimi.it/CMMIMetrics/index.php?id=main.htm
- Software Engineering Institute. (2014). *Carnegie Mellon University*. Retrieved from Capability Maturity Model Integration (CMMI): http://whatis.cmmiinstitute.com/#home
- The Aerospace Corporation. (2011). *Aerospace Software Measurement Standard .* El Segundo, CA: The Aerospace Corporation.
- GAO-14-694. “Healthcare.gov: Ineffective Planning and Oversight Practices Underscore the Need for Improved Contract Management”. July 2014
- HHS ASPE Issue Brief, “Health Insurance Marketplace: Summary Enrollment Report for the Initial Annual Open Enrollment Period”. May 2014
- HHS. “Healthcare.gov Progress and Performance Report”. December 2013