Dr. Richard Turner
turner@software.org
SSTC, Tampa, Florida
June 2006
Founded to facilitate collaboration among our members, who are leaders in Federal/Commercial marketplace

Focused on the challenges of complex system and software development

Committed to the business performance of our members

Structured to provide “neutral ground” and a voice for industry

Beyond Process Improvement. Impacting Member Growth.
Why am I (are you) here?

• To look at systems engineering through the agile lens
• To start (continue?) a dialogue between the proponents of traditional and agile approaches
• To identify some ways that SE can be both more agile and “the cause of agility in others”
• Outline
  – Why bother?
  – Characteristics of Agility
  – Agile System Engineering Processes
  – System Engineering for Agile Software Development
  – Systems Engineering Agile Systems
Why Bother?
Some Software-intensive System Trends

Traditional Development

• Standalone systems
• Stable requirements
• Rqts. determine capabilities
• Control over evolution
• Enough time to keep stable
• Stable jobs
• Failures not critical
• Reductionist systems
• Repeatability-oriented process, maturity models

Current/Future Trends

• Everything connected (maybe)
• Rapid requirements change
• COTS capabilities determine rqts.
• No control over COTS evolution
• Ever-decreasing cycle times
• Outsourced jobs
• Failures critical
• Complex, adaptive, emergent SOSs
• Adaptive process models
The Cast of Characters
(or the Characterization of the Cast?)

- **Agility:** often portrayed as a valid response to
  - Rapid change/increased complexity
  - Emergent, vague, volatile requirements
  - Integration and validation problems late in projects

- **Systems Engineering:** often portrayed as a hold-over from the 50’s and 60’s and part of the problem
  - Rigid and waterfall-based
  - Overly process-bound (MIL-STD-499, MIL-STD-1521)
  - Focused on early correctness; precision over accuracy
Some Software-intensive System Trends

Traditional Development

- Standalone systems
- Stable requirements
- Rqts. determine capabilities
- Control over evolution
- Enough time to keep stable
- Stable jobs
- Failures not critical
- Reductionist systems
- Repeatability-oriented process, maturity models

Current/Future Trends

- Everything connected (maybe)
- Rapid requirements change
- COTS capabilities determine rqts.
- No control over COTS evolution
- Ever-decreasing cycle times
- Outsourced jobs
- Failures critical
- Complex, adaptive, emergent SOSs
- Adaptive process models

Is this really the domain of Traditional SE?
Characteristics of Agility
Characteristics of Agility-1

- Short iterations delivering value
  - Rolling planning horizon; Reality-based iteration planning
  - Goal of each release is a working system
- Focus on value to customer – prioritized requirements
  - Progress measured by operational features
- Change neutrality (design for change)
  - Change is seen as inevitable; ergo “embrace change”
- Team ownership
  - Team has primary responsibility and authority over its own plans and processes
  - Quality/performance is everyone’s responsibility
Characteristics of Agility-2

- Lean attitude (remove no-value-added activities)
  - As little ceremony as necessary; just enough (or just too little) process
  - Decisions delayed until latest feasible time (Toyota approach)

- Test-driven (demonstrable progress)
  - Tests are written before any other artifacts (design, code)
  - Capabilities (requirements) are defined by the tests (empirical evidence) that validate them

- Continuous integration
  - Integration is an ongoing activity
  - Integration and testing as automated as possible
Agility and Process Maturity

- Agility is not “anti-process;” can conform to CMMI (when considered objectively)
  - SSCI currently developing a PIDS for CMMI Lead Appraisers to use in appraising agile projects
- Agile concepts embody “Level 5-ness” by continuously improving/adjusting processes
  - Retrospective/reflection after each iteration
  - Recommendations immediately implemented
  - Measures confirm or contradict changes in next few iterations – not next project
Agile Systems Engineering Processes
• Issue: The assumption (by all stakeholders) that foreknowledge is perfect
  – Perfect requirements decomposed to perfect specifications

• “… for the most part, engineers do not know how to build the systems they are trying to build; it is their job to find out how to build such systems.” *Philip Armour*

• SE process is essentially a learning process
  – Trade studies, requirements analysis, demonstrations
  – Allocation, design evaluation
  – V&V

• Traditional view of SE “V” only gives a limited, “one-time through” chance to learn
• No detailed plan survives the first engagement
• SE C4ISR via spiral OODA loops
  – Observe, Orient, Decide, Act
  – Vs. Requirements, Delay, Surprise
• Concurrent tasking, collaboration technology essential
  – Spanning deep chains of command
    • Customer, LSI, IPT’s (C4ISR), Decision Support, COP
      Refresh, Sensor Fusion, Sensors, Sensor components
• Common strategy essential; micro-planning risky
• Competition, technology, marketplace ISR essential
• Rapid adaptability essential
• Stable development increments also essential

Boehm and Turner, Tutorial, ICSE 2005
Copyright © 2007, Systems and Software Consortium, Inc.
SE C4ISR Via Spiral OODA Loops

**Observe** new/updated objectives, constraints, alternatives
- Usage monitoring
- Competition, technology, marketplace ISR

**Orient** with respect to stakeholders priorities, feasibility, risks
- Risk/Opportunity analysis
- Business case/mission analysis
- Prototypes, models, simulations

**Operate** as current system

**Accept** new system

**Act** on plans, specifications
- Keep development stabilized
- Change impact analysis, preparation for next cycle (mini-OODA loop)

**Decide** on next-cycle capabilities, architecture upgrades, plans
- Stable specifications, COTS upgrades
- Development, integration, V&V, risk management plans
- Feasibility rationale

Life Cycle Architecture Milestone for Cycle

Boehm and Lane, CrossTalk, May 2006
Copyright © 2007, Systems and Software Consortium, Inc.
Systems Engineering for Agile SW Development
• Traditional SE lifecycle seems biased toward software development as sub-component (computer programming of limited functionality rather than software engineering)

• Iterative SW development, isn’t really accounted for in the traditional “big design up front”
  – Synchronizing concurrent engineering efforts
  – COTS HW and SW, Open Source SW and legacy systems cause significant perturbation

• MIL-STD-1521 style reviews are counter-productive
  – Reviews are documentation-based rather than demonstration-based
  – PDR and CDR are traditionally held before software development is begun; much too late for software success
• Delays in starting critical software infrastructure
  – OS, networking, DBMS, transaction processing, …

• Infeasible infrastructure
  – Premature performance requirements (e.g., 1 second)

• Premature hardware selection overconstrains software
  – Can also induce premature COTS commitments

• Waterfall-based progress payments undermine-spiral tasks
  – Develop prototypes or get paid for specifications
Engineering Agile Systems
• NDIA-convened group of industry, government and academia defined top problems in software-intensive systems (which are the majority of what is built)

• One critical finding was:
  – Fundamental system engineering decisions are made without full participation of software engineering.

• Software cannot be relegated to a secondary activity
  – Initial decisions must consider software architecture or they can impact the feasibility of software solutions
• SEI Ultra-Large Systems study metaphor

• Cities are not usually built to a compete set of requirements; cities emerge based on population growth, activities and needs

• Cities are planned at a macro level
  – Infrastructure (power, water), services (police, health), communications, traffic, …
  – Standards established and evolved

• Buildings and other facilities independent
  – Utilize macro definitions, plans and standards
  – Designed and architected for specific or multi-use
Knowing When Not to System Engineer – A multi-platform NC-SOS example

- Customer system-engineers an optimized product line architecture for platform functions
  - Estimates cost savings from reuse
- Customer solicits best-of-breed platform suppliers
  - Contracts with most cost-effective bidders
- Customer discovers that supplier bids are based on product line – incompatible components
  - Too expensive to refit to product line architecture
- Better to risk-manage degree of product line achievability
  - Involve potential suppliers in product line option exploration
• **Scope management** ⇒ Asset based development
  – Solutions need to evolve from user specifications **AND** user specifications need to evolve from candidate solutions.
  – As opposed to getting all the requirements right up front.

• **Process management** ⇒ Rightsize the process
  – Process and instrumentation rigor evolves from light to heavy.
  – As opposed to the entire project’s lifecycle process should be light or heavy depending on the character of the project.

• **Progress management** ⇒ Honest assessments
  – Healthy projects display a sequence of progressions and digressions.
  – As opposed to healthy projects progress to 100% earned value with a monotonically increasing and predictable plan.

• **Quality management** ⇒ Incremental demonstrable results
  – Testing needs to be a 1st class, full lifecycle activity.
  – As opposed to a subordinate, later lifecycle activity.
Final Thoughts

- Traditional system engineering may not fit today’s and tomorrow’s systems
- Agility is a state of mind more than a set of rules
- Process is not the enemy – bad process is
- As systems grow larger and more complex, new ways of dealing with abstraction, concurrency, and uncertainty need to be developed
- The basic goals of systems engineering haven’t changed, just the way they are reached
Questions and Discussion
Backup
SISOS Software Benefits, and Risks

• Accommodating many combinations of options
  – Development speed; integration; cross-system KPP’s
• Accommodating many combinations of systems and contractors
  – Subcontractor specifications, incompatibilities, change management
• Rapid tailoring and upgrade of many combinations of options
  – Version control and synchronous upgrade propagation
• Flexibility, rapid adaptability, incremental development
  – Subcontractor chain increment synchronization; requirements and architecture volatility
• Near-free COTS technology upgrades
  – COTS upgrade synchronization; obsolescence; subcontractor COTS management
• Compound risks
1. Acquisition management and staffing
2. Requirements/architecture feasibility
3. Achievable software schedules
4. Supplier integration
5. Adaptation to rapid change
6. Quality factor achievability and tradeoffs
7. Product integration and electronic upgrade
8. Software COTS and reuse feasibility
9. External interoperability
10. Technology readiness