Agile Systems Engineering Approach to Software Project Development

Chris Ritter, Daniel Hettema, and Steven H. Dam, Ph.D., ESEP, SPEC Innovations, 571-485-7807
chris.ritter@specinnovations.com

October 2013
Overview

- What is “Agile?”
- How is agile development applied to software?
- What is the difference between Systems Engineering and Software Engineering?
- What lessons from Agile make sense to apply to systems engineering?
- Summary
What is “Agile?”

- Time-boxed iterative approach
- Incremental
- Self-organizing, cross-functional teams
- Adaptive planning
- Rapid and flexible response to change

**Manifesto for Agile Software Development**

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

See [http://agilemanifesto.org/](http://agilemanifesto.org/)

We have used many names for this over the years: “build a little, test a little” and “extreme programming” to name a couple.
How is Agile Applied to Software Development?

• Many techniques have been derived (e.g., Kaisen, Scrum)
• But does this kind of software development approach translate well into the DoD/NASA world where software makes life and death decisions?

SPEC’s Agile Software Production Process – Used for Commercial Software Development

**Bi-Weekly “Sprint” Software Production Cycle**
- **Monday – Week 1:** Plan & Evaluate Priorities
- **Tuesday – Week 1:** Analysis & Design
- **Wednesday – Week 1 to Wednesday – Week 2:** Coding (New Features/Bug Fixes) & Unit/Developmental Testing (Local Host)
- **Thursday – Week 2:** Operational Test (Cloud Test Site)
- **Friday AM – Week 2:** Final fixes (if necessary) and review
- **Friday PM – Week 2:** Final OT & Document Updates
- **Friday Late PM – Week 2:** Deploy to Cloud Site
- **New Version Available to Users**

*Innoslate SLOC ≈ 370k*

*We use a small, co-located team who went to school together and have worked together many years*
DoD Software Systems

• Characterized by
  – Millions of lines of code
  – Much of it using older software languages
  – Real-time
  – Standards-based
  – Many, many reviews
  – Many, conflicting “requirements” from many, varied stakeholders

*All driven by a very non-Agile acquisition system*
What Does This Mean to DoD?

• Agile clearly can be applied in non-mission critical activities, for example:
  – Websites
  – Specialized analytical tools
  – “Commercial-like applications” (although it would be better usually to just use COTS)

• It may be applicable to mission-critical software, but only when it’s functional requirements are well-defined and the contracting environment allows for flexibility
What Is the Difference between Systems Engineering and Software Engineering?

• Systems Engineering
  – Capabilities-based
  – Decomposition to many levels
  – Must support all aspects of the lifecycle
  – Hardware SE requires “measure twice, cut once” approach

• Software Engineering
  – Needs to have well-defined functional requirements
    • Verifiable
    • Clear
    • Concise ...
  – Code can be executed quickly
Is There Such a Thing as Agile SE?

• Many people are trying to apply the Agile principles to SE
• Many forums on this subject (see LinkedIn)
• How have we done this?
We Focus on a Middle-Out Process

The middle-out approach has been proven on a variety of projects.

Best Use: "Classical SE"

Best Use: Architecture Development (To-Be)

Best Use: Reverse Engineering (As-Is)

Adapted from EIA-632
Middle-Out Timeline

1. Capture and Analyze Related Artifacts
2. Identify Assumptions
3. Identify Existing/Planned Systems
4. Capture Constraints
5. Develop the Operational Context Diagram
6. Develop Operational Scenarios
7. Derive Functional Behavior
8. Derive Assets
9. Allocate Actions to Assets
10. Prepare Interface Diagrams
11. Define Resources, Error Detection & Recovery
12. Perform Dynamic Analysis
13. Develop Operational Demonstration Master Plan
14. Provide Options
15. Conduct Trade-off Analyses
16. Generate Operational and System Architecture Graphics, Briefings and Reports

Requirements Analysis
Functional Analysis
Synthesis
System Analysis and Control

Originally developed to support ACTDs in the 1990s

Can be executed in as little as two weeks

Time
Agile Architecture Development as Well

1. Capture and Analyze Related Artifacts
2. Identify Assumptions
3. Identify Existing/Planned Systems
4. Capture Constraints
5. Develop the Operational Context Diagram
6. Develop Operational Scenarios
7. Derive Functional Behavior
8. Derive Assets
9. Allocate Actions to Assets
10. Prepare Interface Diagrams
11. Define Resources, Error Detection & Recovery
12. Perform Dynamic Analysis
13. Develop Operational Demonstration Master Plan
14. Prepare Concept of Operations
15. Conduct Trade-off Analyses
16. Generate Operational and System Architecture Graphics, Briefings and Reports

Time

Fits with DoD requirements
What Lessons from Agile Make Sense to Apply to Systems Engineering?

• Individuals and interactions over processes and tools
  – However, you need to apply a process for repeatability and tools to capture the information

• Working software (design) over comprehensive documentation
  – Implies MBSE with simulation

• Customer collaboration over contract negotiation
  – Need to work better as a Government-Contractor team

• Responding to change over following a plan
  – Need flexibility in the SOW and deliverables to make this work
  – Implies CPFF or T&M?
Summary

• Agile is the latest buzzword for something we have been trying to do for decades
• The problem really comes from people being inflexible and suspicious of each other
• A measure of trust is needed to actually improve the situation