Systems Engineering for Software Engineers
Part 2

Developed and presented by
Richard E. (Dick) Fairley, PhD, CSDP
Principal Associate
Software and Systems Engineering Associates (S2EA)
d.fairley@computer.org

Tutorial Agenda

Part 1: An overview of systems engineering

Part 2: Systems engineering foundations and a systems/software development strategy

Part 2 topics

- System engineering foundations
- System engineering and software engineering terminology
- Systems and software development strategies
- An integrated systems/software development strategy

Systems engineering foundations

Foundations for systems engineering include:

- Systems thinking
- Systems science
- SEBOK: The systems engineering body of knowledge www.sebok.org
- ISO/IEC/IEEE Standard 15288:2015

Systems Thinking

- Systems thinking is a *mindset* that involves embracing a holistic view of a complex system to understand the system elements and their interactions by focusing on the context in which the system exists, the contextual relationships among the system elements, and contextual relationships with other systems.

Key Words in Systems Thinking

- Holistic: the parts of a system and their interactions can be best understood (perhaps only understood) by considering the entire system
- Context: the environment of a system provides the boundaries and interfaces for the system, which support understanding of required and desired system behavior
- Emergence: new, unanticipated system behaviors may emerge as system elements are added or modified
  - The new behaviors may be positive or negative
- Unintended consequences: emergence may result in unforeseen, often undesirable, effects within the system or on the environment of the system

Systems Science*

- Farlex* offers the following definition of systems science:
- Systems science is an interdisciplinary field of science that studies the nature of complex systems in nature, society, and science. It aims to develop interdisciplinary foundations, which are applicable in a variety of areas, such as engineering, biology, medicine and social sciences. (Farlex 2012).

* http://www.thefreedictionary.com/

* See Part 2 of the SEBoK Guide

Foundations for systems engineering

Foundations for systems engineering include:

- Systems thinking
- Systems science
- SEBOK: The systems engineering body of knowledge www.sebok.org
- ISO/IEC/IEEE Standard 15288 (2015)

SEBoK: a guide to the systems engineering body of knowledge: sebokwiki.org

SEBoK has 7 parts

- Part 1: SEBoK Introduction
- Part 2: Foundations of Systems Engineering
- Part 3: SE and Management
- Part 4: Applications of Systems Engineering
- Part 5: Enabling Systems Engineering
- Part 6: Related Disciplines
- Part 7: SE Implementation Examples

SEBoK Structure

- SEBOK has 7 parts
- Each part contains knowledge areas (KAs)
- Each KA includes topics
  - 25 KAs; 114 topics
- Each topic includes an overview and references for further readings on the topic

SEBoK Part 6 Related Disciplines

Part 6: Related Disciplines
- Software engineering
- Project management
- Industrial engineering
- Procurement/acquisition
- Specialty engineering *

Part 7: Vignettes and Case Studies

* "specialty engineering" is a systems engineering term

SEBoK Part 6 Specialty Engineering

- Reliability, Availability, and Maintainability
- Human Systems Integration
- Safety Engineering
- Security Engineering
- System Assurance
- EM Interference/EM Compatibility
  - EM: electromagnetic
- Resilience Engineering
- Manufacturability and Producibility
- Affordability
- Environmental Engineering

The SEBoK wiki

- The SEBoK wiki is built on the MediaWiki open source platform
  - MediaWiki was developed as the platform to implement wikipedia
- The SEBoK wiki has embedded links to facilitate navigation within the document
- The wiki also facilitates periodic modifications and updates

A SWEBOK wiki is being developed by the Computer Society using the SEBoK wiki infrastructure

Foundations for systems engineering

Foundations for systems engineering include:

- System thinking
- Systems science
- SEBOK: The systems engineering body of knowledge www.sebok.org
- ISO/IEC/IEEE Standard 15288:2015

ISO/IEC/IEEE Standard 15288:2015
Systems and software engineering — System life cycle processes

- 15288 establishes a common framework of process descriptions for describing the life cycle of systems created by humans.
- It defines a set of processes and associated terminology from an engineering viewpoint. These processes can be applied at any level in the hierarchy of a system's structure.
- When a system element is software, the software life cycle processes in ISO/IEC/IEEE 12207:2008 may be used to implement that system element.

ISO/IEC/IEEE Standard 15288:2015

From the ISO catalog*

"ISO/IEC/IEEE 15288:2015 concerns those systems that are man-made and may be configured with one or more of the following system elements: hardware, software, data, humans, processes, procedures, facilities, materials, and naturally occurring elements."

* http://www.iso.org/iso/home/store/catalogue_tccatalogue_detail.htm?csnumber=63711

15288 System Life Cycle Processes

Agreement Processes
- Acquisition Process
- Supply Process

Organizational Project-Enabling Processes
- Lifecycle Model Management
- Infrastructure Management
- Portfolio Management
- Human Resource Management
- Quality Management
- Knowledge Management

Project Processes
- Project Planning
- Process Assessment & Control
- Decision Management
- Risk Management
- Configuration Management
- Information Management
- Measurement Process
- Quality Assurance Process

Technical Processes
See next slide

30 processes grouped into 4 areas

15288: 14 Technical Processes

- Business or Mission Analysis
- Stakeholder Needs & Requirements Definition
- System Requirements Definition
- Architecture Definition
- Design Definition
- System Analysis
- Implementation of hardware and software elements
- Integration
- Verification
- Transition
- Validation
- Operation
- Maintenance
- Disposal

The 15288 Implementation Process

- "The purpose of the Implementation Process is to realize a specified system element"
  - Systems engineers do not implement system components

    Although some SEs may do some implementation
- Software development is (just another) systems engineering implementation process
  - ISO/IEC/IEEE Standard 12207:2008 describes software development processes

ISO/IEC/IEEE Standard 12207:2008

- 12207:2008 is the software specialization of the systems engineering processes in 15288:2015
- 12207 is the umbrella standard for the ISO and IEEE software engineering standards
  - all IEEE SwE standards are (or should be) harmonized with 12207

12207 is the only system-element standard that is closely integrated with 15288

S2EA

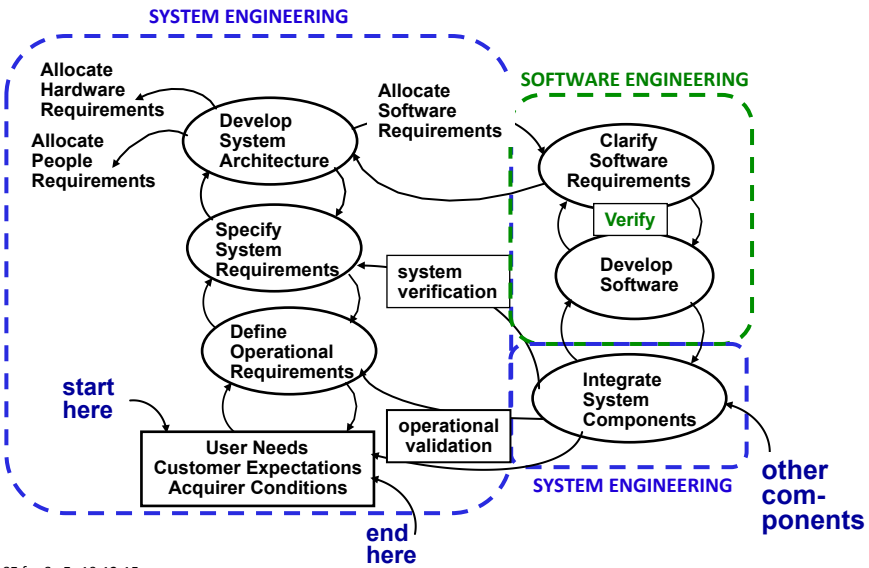
A problem

- Software modules are system elements
  - One kind of element among others
    - Including hardware components and manual operations
- Yet software is an element of most, if not all, other components and provides the interfaces among them and interfaces to the environment

Many systems engineers view software like any other kind of system element
To be implemented by software engineers

S2EA

A software engineer's view of systems engineering and software engineering



A software engineer's view of systems engineering (2)

1. System requirements allocated to software are often vague and ambiguous
2. Interfaces to hardware components are not documented
   - Or if documented, they change frequently and without notification
3. Technical interchange meetings with other engineers (and vendors and subcontractors) are not established
4. System engineers are not available for consultation

A system engineer's view of software engineers

1. Software engineers are often too narrowly focused
   - And don't care about the "bigger" picture
2. Software engineers seem to focus on the development processes they use more than the products they develop
3. Software engineers seem to speak a different language than do other engineers
   - And have a different approach to problem solving
   - ("they aren't real engineers")
4. Software engineers think systems engineers use outdated processes
   - Perhaps so, but mostly because software engineers don't understand the constraints hardware development places on development processes

What systems engineers do (or should do) during component implementation (1)

- They interact (or should interact) with component specialists to:
  - to revise the system architecture, as needed
  - perform trade studies for components
  - make tradeoffs among hardware, software, and people requirements
- In addition, systems engineers:
  - Develop(or should develop) system integration and system verification plans
  - Develop (or should develop) transition and validation plans

product verification is done in the development (factory) environment
product validation is done in the users' (operational) environment

What systems engineers do (or should do) during component implementation (2)

- In some cases, systems engineers do system analysis, system design, and requirements allocation
  - *And then disappear*
  - Development of interface details and interactions among system components are left to others
  - Development and implementation of system integration and system verification plans, and transition and validation plans *are left to others*

In many cases, system engineers are more involved in the "front end" processes than in the "middle" and "back end" processes

What do software engineers do?

When systems engineering is inadequate:

Software engineers will guess at what they think users and customers want and need

Part 2 topics

- System engineering foundations
- System engineering and software engineering terminology
- Systems and software development strategies
- An integrated systems/software development strategy

Systems engineering terminology

- Systems engineers, like others, use terms with special meaning
  - and software engineers sometimes use the same terms with different meanings
  - For example:
    - System element
    - Hierarchy
    - Model-based
    - Performance
    - Implementation
    - Prototyping
    - Specialty engineering
    - Verification and validation
    - System-of-systems

System element

- Systems engineers say a *system element* is part of a system that has allocated requirements, connections to other elements, and provides an element of functionality and behavior to the overall system
  - System elements include hardware, software, and manual operations provided by humans
- Software engineers say a system element is a software module, process, procedure, function, subroutine, macro, etc

Hierarchy

- Systems engineers use *hierarchy* to denote the decomposition of a system into elements and sub-elements
  - Because each system element and sub-element is a hardware element
- Software engineers use *shallow* hierarchy to denote
  - Mapping of software interfaces to hierarchical hardware elements
  - And specialization and generalization
    - i.e. inheritance

Model-based systems engineering

- System engineers use *model-based* to mean Model-based Systems Engineering (MBSE), which is concerned with creating and manipulating iconic system models (e.g., SysML diagrams)
  - Rather than using requirements and design documents to communicate information among engineers and engineering teams
- A goal of MBSE is to develop software-based simulations generated from the system models
  - e.g., executable SysML diagrams

Model-based software engineering

- Software engineers use *model-based* to mean Model Based Software Engineering (MBSwE)
  - Which involves domain analysis and application engineering
  - Domain engineering develops reusable software assets
  - Application engineering uses the software assets to develop software products
  - MBSwE is used in software product line engineering

Performance

- Systems engineers use the term "performance" to mean *total effectiveness of a system*
  - As in "the performance envelope" of the system
    - e.g., reliability, availability, responsiveness, robustness
- Software engineers use the term "performance" to mean:
  - response time to queries and commands
  - Speed of data processing throughput

Implementation

- Systems engineers use implementation to mean development and verification of a system element based on allocated requirements
- Software engineers use *implementation* to mean software construction (detailed design, coding, unit testing, integration of software modules)

Prototyping

- In systems engineering, a *prototype* is usually a first fully functioning version of a system
- In software, a prototype is a mockup of some part of the software that is to be built:
  - To simulate a user interface
  - Or to explore a technical issue

Specialty Engineering

- *Specialty engineering* includes engineering disciplines that are not usually in the mainstream of product development
  - Examples: system safety, physical security, EMI
- Specialists may be called on as needed
  - They may be members of a "specialty group"
  - Or they may be consultants
- Software engineers use the term "supporting processes" to denote those processes that are not in the mainstream of software development
  - Examples: CM, QA, IV&V

Verification and Validation

- Software engineers and systems engineers agree that "verification" is concerned with determining the extent to which a product or a product element satisfies its allocated requirements, based on objective evidence
- Software engineers and systems engineers agree that "validation" is concerned with determining the extent to which a product satisfies user needs and customer expectation when used in the intended ways in the intended environment

Validation – except that

- Except that, systems engineers stress the need for *objective evidence* during the validation process, which may be possible based on objective hardware measures
  - Software engineers often rely on *subjective impressions* of users and customer to validate software

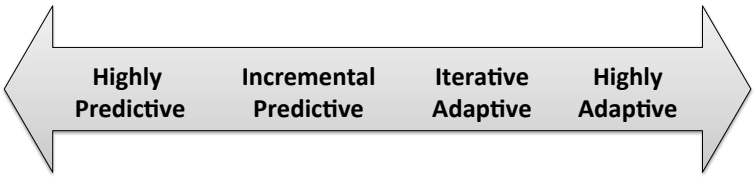
Systems-of-systems

- A *system-of-systems* is a collection of systems that are integrated to work together
  - And were not initially designed to be integrated
- For example:
  - consolidating various organizational IT elements into an enterprise IT system
  - Or consolidating a group of customer applications into a product line
    - With a base product and application extensions
  - Other examples?

Part 2 topics

- System engineering foundations
- System engineering terminology
- System and software development strategies
- An integrated system/software development strategy

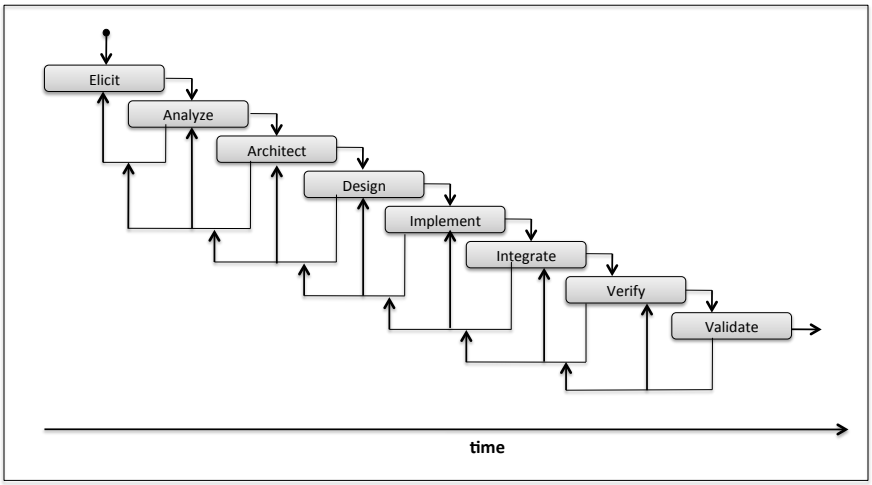
The product development continuum



- Highly Predictive (HPDM): linear, one-pass phases
- Incremental Predictive (IPDM): piecewise linear phases
- Iterative-Adaptive (IADM): iterative phases
- Highly Adaptive (HADM): highly iterative phases

see the Software Extension to the *PMBOK® Guide Fifth Edition*
http://marketplace.pmi.org/Pages/ProductDetail.aspx?GMProduct=00101457501

A Highly Predictive Development Model (HPDM)

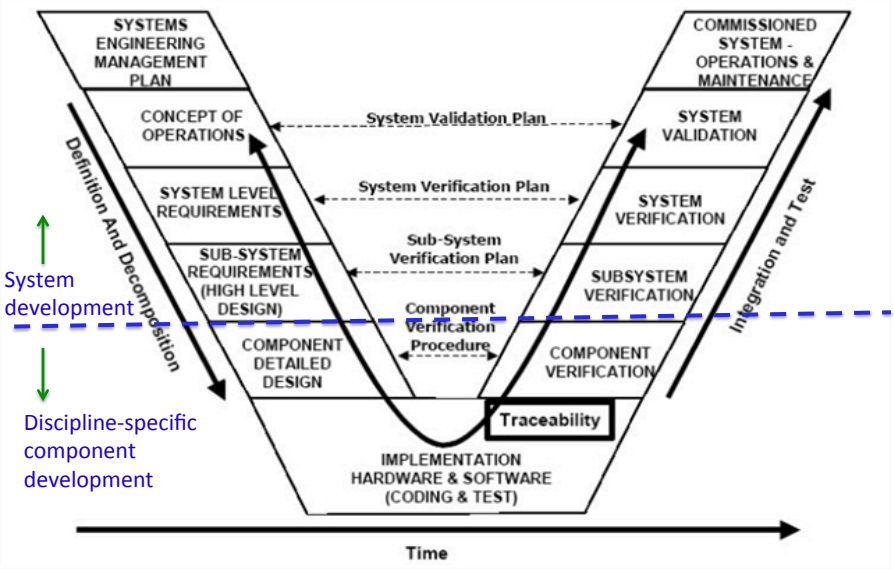


The HPDM goals

- To correctly and completely specify the system requirements during the initiation and planning phases of a systems project
- To initially prepare accurate estimates of schedule, resources, technology, and cost based on the requirements
- To develop a suitable system architecture before proceeding to system implementation
- To correctly and completely finish each phase of system development before proceeding to the next phase

S2EA

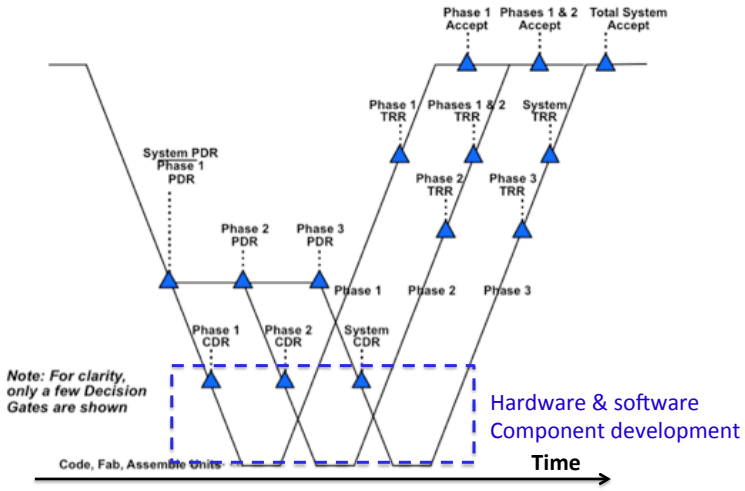
The Vee HPDM



http://www.engineering.com/portals/0/Blogfiles/Systems_Engineering_V_diagram.jpg

S2EA

Incremental Vee HPDM



http://sebokwiki.org/w/images/4/49/KF_IncrementalDevelopment_Multiple.png

Predictive and incremental predictive system development

- Most system engineers use predictive and incremental predictive models for system development
  - Because of the nature of physical hardware
  - With emphasis on hierarchical analysis and design prior to procurement and fabrication of hardware components
    - Because hardware is more difficult to later modify than is software

Hardware analysis and design

- Upfront hardware analysis and design is facilitated by the physical parameters of hardware
  - Mass
  - Volume
  - Dimensions
  - Power
  - EMI/EMC
  - Shock & vibration
  - Heat generation & dissipation

See the ATM example in Part 1

Note

- System engineers "try to get it right" before committing to hardware fabrication and procurement
- As a consequence, they place strong emphasis on preliminary design reviews (PDRs) and detailed design reviews (CDRs)
- Which may not be appropriate reviews for software
  - Software "PDRs" are reviews of the architecture
    - They are the "critical" software reviews
  - Software CDRs, if they occur, are part of detailed design and construction

An issue for HPDM approaches

Problems created first are detected last

When they tend to be difficult and costly to fix

Adaptive software development

- Most software developers prefer adaptive* development processes
  - to iteratively add increasing incremental capabilities to the software
  - to provide frequent objective evidence of progress
  - And to permit incremental changes, as needed

* Adaptive software development is "agile" to the extent that the development model being used satisfies the 10 attributes of agility

See the supplemental material at the end of these slides

An issue

How can predictive hardware development and adaptive software development be meshed?

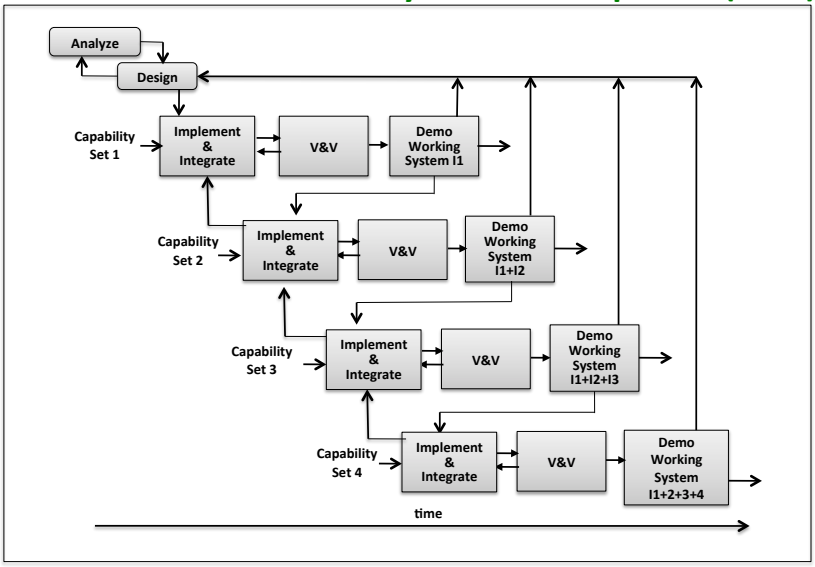
S2EA

Part 2 topics

- System engineering foundations
- System engineering terminology
- System and software development strategies
- An integrated system/software development strategy

S2EA

Incremental-Predictive System Development (IPSD)



Using IPSD

An IPSD approach is appropriate when:

- System requirements can be mostly defined initially
  - And allocated to prioritized capability sets
- Some flexibility in managing prioritized requirements will be permitted
  - plus dynamic allocation (and reallocation) of resources and schedule to increments
- Periodic V&V and stakeholder involvement provides feedback
  - as a basis for assessing progress
  - and for adding, deleting, and revising capabilities

Some IPSD Capability Prioritization Criteria

- Establish the architectural skeleton first and incrementally add capabilities OR
- Allocate the system requirements to increments of growing system subsets some of which are to be periodic delivered for user evaluation or subset operational use OR
- Establish interfaces to external components first the incrementally add internal components OR
- Establish interfaces to users and operators first the incrementally add internal components and external interfaces OR
- Incrementally incorporate components to be reused
  - Both hardware and software

Internal IPSD development processes

- Internal implementation processes may be linear, incremental, or iterative
  - Internal processes may vary among increments
  - And may differ for hardware and software
    - different levels of requirements volatility or uncertainty
    - different organizational units or contractors
- *Hardware interfaces can be emulated in software initially and incrementally replaced as hardware components become available*
- Incremental V&V may be a separate process or may occur as part of the implementation activity

Note

Incremental Predictive System Development is compatible with Model-Based Systems Engineering

- Hardware interfaces and dummy hardware components are initially emulated in software
- Software is developed incremental and the emulated hardware components are replaced with real hardware components as they become available
- System performance parameters can be incrementally evaluated as the system evolves

Tutorial Wrap-up

Part 1 topics

- Introductions
- Systems classifications
- Some examples of systems
- The synergy of software engineering and systems engineering
- Systems engineering notations
- The Systems Engineering Management Plan (SEMP)

Part 2 topics

- System engineering foundations
- System engineering and software engineering terminology
- Systems and software development strategies
- An integrated systems/software development strategy

Tutorial Wrap-up (3)

Any final questions, comments, or concerns?

Supplemental Material

- Adaptive software development is “agile” to the extent that the development process being used satisfies the 10 tenets of agility

Attributes of Agility for Adaptive SDLCs

1. Increments of working deliverable software are produced on a periodic basis
2. Adaptive iteration cycles are of the same duration (i.e., are "time boxed") for a project but some cycles may be of longer or shorter duration by exception
   - durations of the time-boxed iteration cycles may be weekly, bi-weekly, or monthly but not more than monthly
3. Increments of working deliverable software are not necessarily produced by each iteration
   - increments and iterations are distinct
4. Requirements, design, and the software product emerge as the project evolves

Attributes of Agility (2)

5. A representative customer, customer's representative, and/or knowledgeable user is involved on a *continuing basis*
   - involvement includes observing periodic demonstrations of working, deliverable software at the end of each iterative development cycle
   - In addition, a representative customer or customer's representative *accepts shared responsibility* for further product development based on demonstrations of working deliverable software
     - and accept responsibility for managing the constraints on project scope (schedule, budget, and resources)

Notes

- The customer representative *must commit to*:
  a. Being continuously available
  b. Providing updates to the requirements based on periodic demonstrations of the incrementally evolving product *that involve the customer and other key stakeholders*
  c. Discussing proposed changes to the requirements with the software developers to assess technical risk and the impact of proposed changes on the product, schedule, and needed resources
  d. Accepting responsibility for maintain a balance among schedule, budget, and requirements to be implemented

Attributes of Agility (3)

6. Adaptive software development teams are small (i.e., 10 or fewer members) and are self-organizing
   - large projects include multiple small teams
7. Short daily standup meeting, retrospective meetings, and planning meetings are used
8. Each software development team is cross-functional
   - includes the combination of skills needed to accomplish the work activities
   - functional experts may be involved occasionally as needed

Attributes of Agility (4)

9. All members of each software development team are assigned to one project at a time
10. Overtime is *not required*
    - Except on rare occasions of limited durations
    - Software developers should be able to maintain a steady pace indefinitely
        - Without burnout
        - And without sacrificing their personal lives

Projects are "agile" to the extent that they observe these 10 attributes of agility