Incremental Life Cycle Model

Contributing Author: *Kevin Forsberg*

There are a large number of life cycle process models. As discussed in the System Life Cycle Process Drivers and Choices article pre-specified single-step is like traditional or waterfall with fixed requirements; pre-specified, multi-step will develop an early initial operational capability and then follow that by several pre-planned product improvement. The next 3 are incremental and evolutionary and range from rapid fielding to maturing technology to emergent development. We have grouped these models into three major categories: (1) primarily pre-specified single-step or multistep, also known as traditional or sequential processes; (2) evolutionary sequential (or the Vee model); and (3) evolutionary opportunistic and evolutionary concurrent (or incremental agile). The concurrent processes are known by many names: the agile unified process (formerly the Rational Unified Process), the spiral models and include some that are primarily interpersonal and unconstrained processes (e.g., agile development, Scrum, extreme programming (XP), dynamic system development methods, and innovation-based processes).

This article discusses evolutionary opportunistic and evolutionary concurrent, the third category listed above. While there are a number of different models describing the project environment, the spiral model and the Vee Model have become the dominant approaches to visualizing the development process. Both the Vee and the spiral are useful models that emphasize different aspects of a system life cycle.

General implications of using incremental models for

system design and development are discussed below. For a more specific understanding of how this life cycle model impacts systems engineering activities, please see the other knowledge areas (KAs) in Part 3. This article is focused on the use of incremental life cycle process models in systems engineering. (See Systems Engineering and Software Engineering in Part 6 for more information on life cycle implications in software engineering.)

□

Contents

Evolutionary and Incremental Development
- Overview of the Evolutionary Approach
- Overview of the Incremental Approach

Iterative Software Development Process Models
- Overview of Iterative-Development Process Models

The Incremental-Build Model
- The Role of Prototyping in Software Development
- Life Cycle Sustainment of Software
- Retirement of Software

Primarily Evolutionary and Concurrent Processes: The Incremental Commitment Spiral Model
- Overview of the Incremental Commitment Spiral Model
- Other Views of the Incremental Commitment Spiral Model
- Underlying ICSM Principles
- Model Experience to Date

Agile and Lean Processes
- Scrum
  - Architected Agile Methods
- Agile Practices and Principles
- Lean Systems Engineering and Development
  - Origins
  - Principles

References
- Works Cited
- Primary References
- Additional References

Evolutionary and Incremental Development

Overview of the Evolutionary Approach

A specific methodology called evolutionary development is common in research and development (R&D) environments in both the government and commercial sector. Figure 1 illustrates this approach, which was used in the evolution of the high temperature tiles for the NASA Space Shuttle (Forsberg 1995). In the evolutionary approach, the end state of each phase of development is unknown, though the goal is for each phase to result in some sort of useful product.

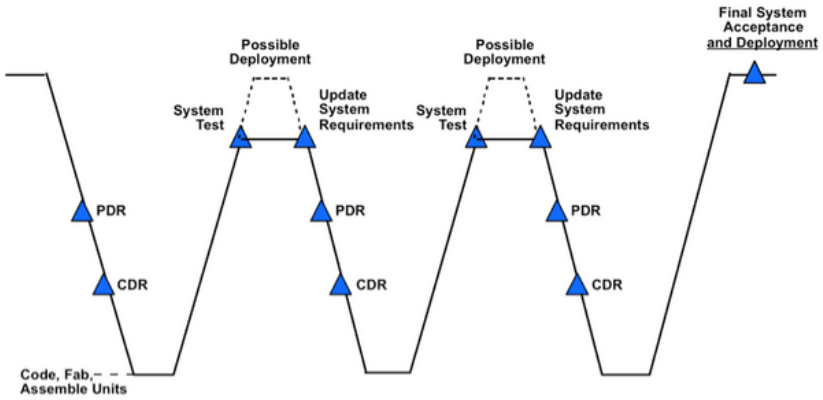

Figure 1. Evolutionary Generic Model (Forsberg, Mooz, Cotterman 2005). Reprinted with permission of John Wiley & Sons, Inc. All other rights are reserved by the copyright owner.

The real-world development environment is complex and difficult to map because many different project cycles are underway simultaneously.

Overview of the Incremental Approach

Incremental development methods have been in use since the 1960s (and perhaps earlier). They allow a project to provide an initial capability followed by successive deliveries to reach the desired system-of-interest (SoI).

The incremental approach, shown in Figure 2, is used when:

- rapid exploration and implementation of part of the system is desired;
- the requirements are unclear from the beginning;

- funding is constrained;
- the customer wishes to hold the SoI open to the possibility of inserting new technology at a later time; and/or
- experimentation is required to develop successive prototype (glossary) versions.

The attributes that distinguish incremental from the single-pass, plan-driven approach are velocity and adaptability.

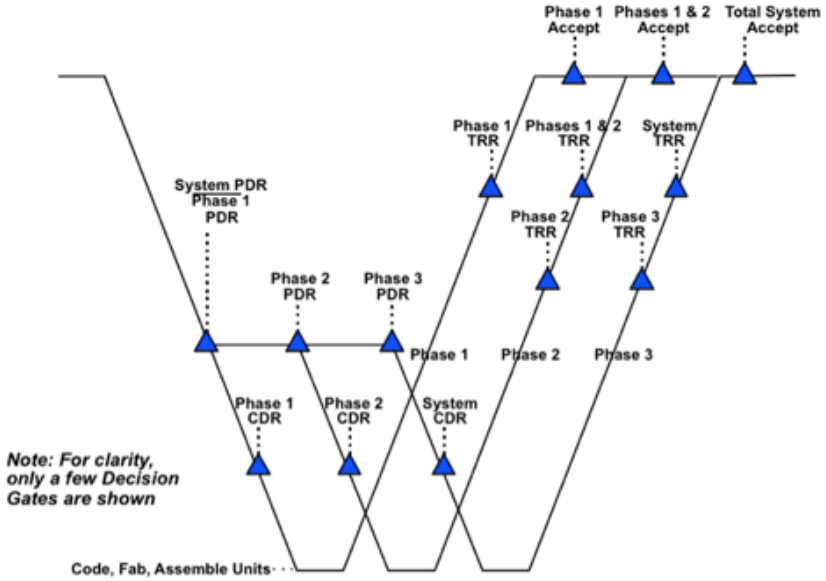

Figure 2. Incremental Development with Multiple Deliveries (Forsberg, Mooz, and Cotterman 2005).

Incremental development may also be “plan-driven” in nature if the requirements are known early on in the life cycle. The development of the functionality is performed incrementally to allow for insertion of the latest technology or for potential changes in needs or requirements. Incremental development also imposes constraints. The example shown in Figure 3 uses the increments to develop high-risk subsystems (or components) early, but the system cannot function until all increments are complete.

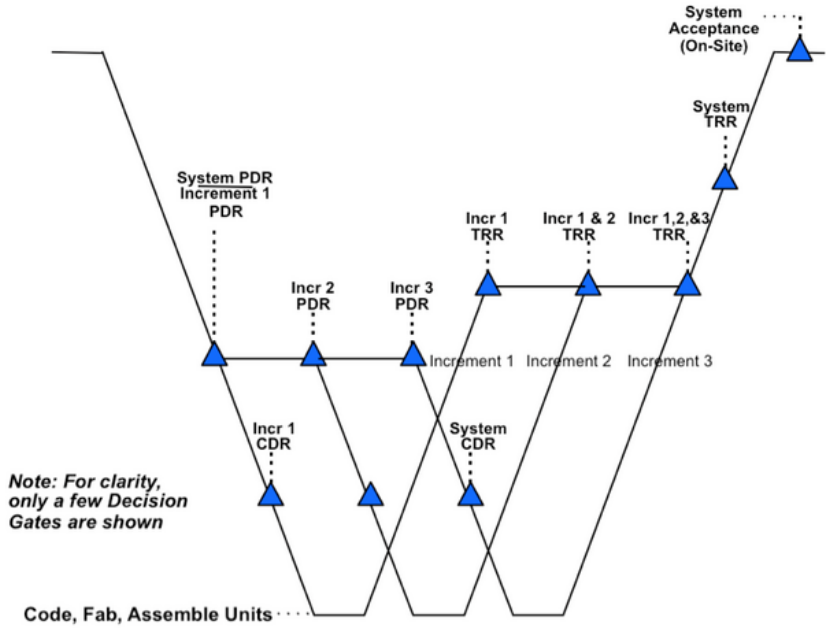

Figure 3. Incremental Development with a Single Delivery (Forsberg, Mooz, Cotterman 2005). Reprinted with permission of John Wiley & Sons Inc. All other rights are reserved by the copyright owner.

Advise this figure and section be moved to the Case Study section of the SEBoK.--- Figure 4 shows the applied research era for the development of the space shuttle Orbiter and illustrates multi-levels of simultaneous development, trade-studies, and ultimately, implementation.

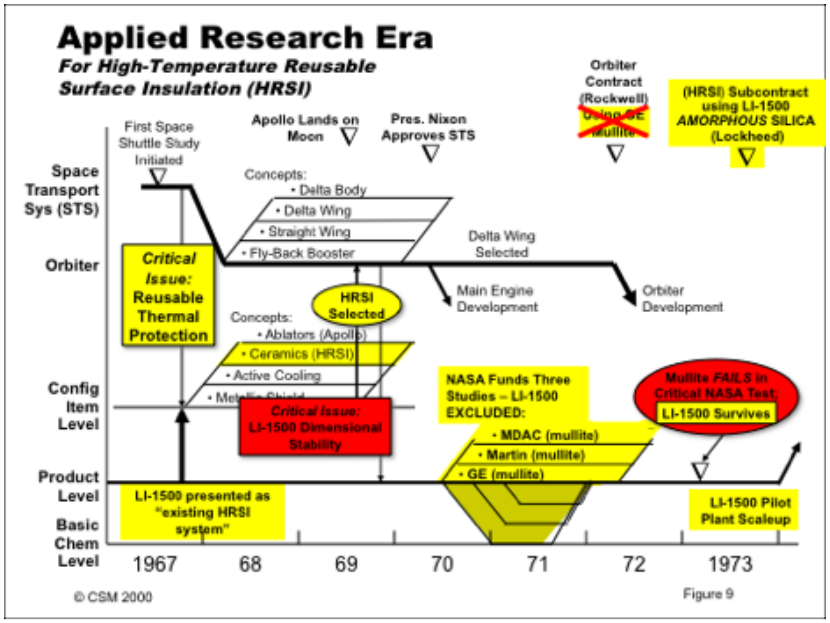

Figure 4. Evolution of Components and Orbiter Subsystems (including space shuttle tiles) During Creation of a Large "Single-Pass" Project (Forsberg 1995). Reprinted with permission of Kevin Forsberg. All other rights are reserved by the copyright owner.

Advise this information below be moved to Part 6, unless

it is redundant or out of date.---

Iterative Software Development Process Models

Software is a flexible and malleable medium which facilitates iterative analysis, design, construction, verification, and validation to a greater degree than is usually possible for the purely physical components of a system. Each repetition of an iterative development model adds material (code) to the growing software base; the expanded code base is tested, reworked as necessary, and demonstrated to satisfy the requirements for the baseline.

Process models for software development support iterative development on cycles of various lengths. Table 1 lists three iterative software development models which are presented in more detail below, as well as the aspects of software development that are emphasized by those models.

Table 1. Primary Emphases of Three Iterative Software Development Models. (SEBoK Original)

Iterative Model	Emphasis
Incremental-build	Iterative implementation-verification-validations-demonstration cycles
Spiral	Iterative risk-based analysis of alternative approaches and evaluation of outcomes
Agile	Iterative evolution of requirements and code

Please note that the information below is focused specifically on the utilization of different life cycle models for software systems. In order to better understand the interactions between software engineering (SwE) and systems engineering (SE), please see the Systems Engineering and Software Engineering KA in Part 6.

Overview of Iterative-Development Process Models

Developing and modifying software involves creative processes that are subject to many external and changeable forces. Long experience has shown that it is impossible to “get it right” the first time, and that iterative development processes are preferable to linear, sequential development process models, such as the well-known Waterfall model. In iterative development, each cycle of the iteration subsumes the software of the

previous iteration and adds new capabilities to the evolving product to create an expanded version of the software. Iterative development processes provide the following advantages:

- Continuous integration, verification, and validation of the evolving product;
- Frequent demonstrations of progress;
- Early detection of defects;
- Early warning of process problems;
- Systematic incorporation of the inevitable rework that occurs in software development; and
- Early delivery of subset capabilities (if desired).

Iterative development takes many forms in SwE, including the following:

- An incremental-build process, which is used to produce periodic (typically weekly) builds of increasing product capabilities;
- Agile development, which is used to closely involve a prototypical customer in an iterative process that may repeat on a daily basis; and
- The spiral model, which is used to confront and mitigate risk factors encountered in developing the successive versions of a product.

The Incremental-Build Model

The incremental-build model is a build-test-demonstrated model of iterative cycles in which frequent demonstrations of progress, verification, and validation of work-to-date are emphasized. The model is based on stable requirements and a software architectural specification. Each build adds new capabilities to the incrementally growing product. The process ends when the final version is verified, validated, demonstrated, and accepted by the customer.

Table 2 lists some partitioning criteria for incremental development into incremental build units of (typically) one calendar week each. The increments and the number of developers available to work on the project determine the number of features that can be included in each incremental build. This, in turn, determines the overall schedule.

Table 2. Some partitioning criteria for incremental

builds (Fairley 2009). Reprinted with permission of the IEEE Computer Society and John Wiley & Sons Inc. All other rights are reserved by the copyright owner.

Kind of System	Partitioning Criteria
Application package	Priority of features
Safety-critical systems	Safety features first; prioritized others follow
User-intensive systems	User interface first; prioritized others follow
System software	Kernel first; prioritized utilities follow

Figure 5 illustrates the details of the build-verify-validate-demonstrate cycles in the incremental build process. Each build includes detailed design, coding, integration, review, and testing done by the developers. In cases where code is to be reused without modification, some or all of an incremental build may consist of review, integration, and testing of the base code augmented with the reused code. It is important to note that development of an increment may result in reworking previous components developed for integration to fix defects.

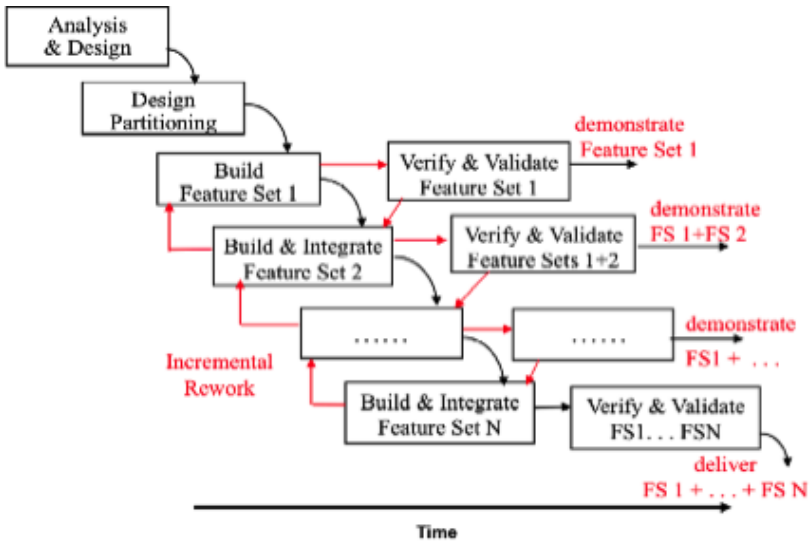

Figure 5. Incremental Build-Verify-Validate-Demonstrate Cycles (Fairley 2009). Reprinted with permission of the IEEE Computer Society and John Wiley & Sons Inc. All other rights are reserved by the copyright owner.

Incremental verification, validation, and demonstration, as illustrated in Figure 5, overcome two of the major problems of a waterfall approach by:

- exposing problems early so they can be corrected as they occur; and
- incorporating minor in-scope changes to requirements that occur as a result of incremental demonstrations

in subsequent builds.

Figure 5 also illustrates that it may be possible to overlap successive builds of the product. It may be possible, for example, to start a detailed design of the next version while the present version is being validated.

Three factors determine the degree of overlap that can be achieved:

1. Availability of personnel;
2. Adequate progress on the previous version; and
3. The risk of significant rework on the next overlapped build because of changes to the previous in-progress build.

The incremental build process generally works well with small teams, but can be scaled up for larger projects.

A significant advantage of an incremental build process is that features built first are verified, validated, and demonstrated most frequently because subsequent builds incorporate the features of the earlier iterations. In building the software to control a nuclear reactor, for example, the emergency shutdown software could be built first, as it would then be verified and validated in conjunction with the features of each successive build.

In summary, the incremental build model, like all iterative models, provides the advantages of continuous integration and validation of the evolving product, frequent demonstrations of progress, early warning of problems, early delivery of subset capabilities, and systematic incorporation of the inevitable rework that occurs in software development.

The Role of Prototyping in Software Development

In SwE, a prototype is a mock-up of the desired functionality of some part of the system. This is in contrast to physical systems, where a prototype is usually the first fully functional version of a system (Fairley 2009, 74).

In the past, incorporating prototype software into production systems has created many problems. Prototyping is a useful technique that should be employed as appropriate; however, prototyping is *not* a process model for software development. When building

a software prototype, the knowledge gained through the development of the prototype is beneficial to the program; however, the prototype code may not be used in the deliverable version of the system. In many cases, it is more efficient and more effective to build the production code from scratch using the knowledge gained by prototyping than to re-engineer the existing code.

Life Cycle Sustainment of Software

Software, like all systems, requires sustainment efforts to enhance capabilities, adapt to new environments, and correct defects. The primary distinction for software is that sustainment efforts change the software; unlike physical entities, software components do not have to be replaced because of physical wear and tear. Changing the software requires re-verification and re-validation, which may involve extensive regression testing to determine that the change has the desired effect and has not altered other aspects of functionality or behavior.

Retirement of Software

Useful software is rarely retired; however, software that is useful often experiences many upgrades during its lifetime. A later version may bear little resemblance to the initial release. In some cases, software that ran in a former operational environment is executed on hardware emulators that provide a virtual machine on newer hardware. In other cases, a major enhancement may replace and rename an older version of the software, but the enhanced version provides all of the capabilities of the previous software in a compatible manner. Sometimes, however, a newer version of software may fail to provide compatibility with the older version, which necessitates other changes to a system.

Primarily Evolutionary and Concurrent Processes: The Incremental Commitment Spiral Model

Overview of the Incremental Commitment Spiral Model

A view of the Incremental Commitment Spiral Model (ICSM) is shown in Figure 6.

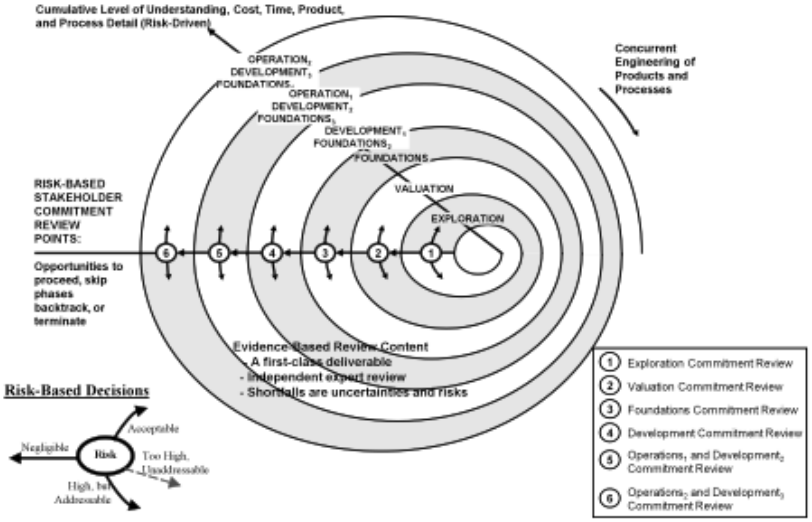

Figure 6. The Incremental Commitment Spiral Model (ICSM) (Pew and Mavor 2007). Reprinted with permission by the National Academy of Sciences, Courtesy of National Academies Press, Washington, D.C. All other rights are reserved by the copyright owner.

In the ICSM, each spiral addresses requirements and solutions concurrently, rather than sequentially, as well as products and processes, hardware, software, human factors aspects, and business case analyses of alternative product configurations or product line investments. The stakeholders consider the risks and risk mitigation plans and decide on a course of action. If the risks are acceptable and covered by risk mitigation plans, the project proceeds into the next spiral.

The development spirals after the first development commitment review follow the three-team incremental development approach for achieving both agility and assurance shown and discussed in Figure 2, "Evolutionary-Concurrent Rapid Change Handling and High Assurance" of System Life Cycle Process Drivers and Choices.

Other Views of the Incremental Commitment Spiral Model

Figure 7 presents an updated view of the ICSM life cycle process recommended in the National Research Council *Human-System Integration in the System Development Process* study (Pew and Mavor 2007). It was called the Incremental Commitment Model (ICM) in the study. The ICSM builds on the strengths of current process models, such as early verification and validation concepts in the

Vee model, concurrency concepts in the concurrent engineering model, lighter-weight concepts in the agile and lean models, risk-driven concepts in the spiral model, the phases and anchor points in the rational unified process (RUP) (Kruchten 1999; Boehm 1996), and recent extensions of the spiral model to address systems of systems (SoS) capability acquisition (Boehm and Lane 2007).

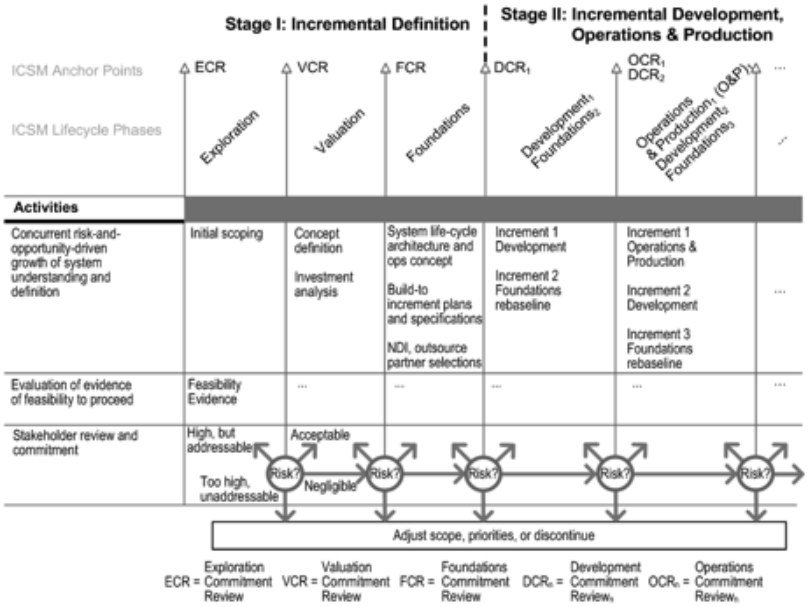

Figure 7. Phased View of the Generic Incremental Commitment Spiral Model Process (Pew and Mavor 2007).

The top row of activities in Figure 7 indicates that a number of system aspects are being concurrently engineered at an increasing level of understanding, definition, and development. The most significant of these aspects are shown in Figure 8, an extension of a similar *"hump diagram"* view of concurrently engineered software activities developed as part of the RUP (Kruchten 1999).

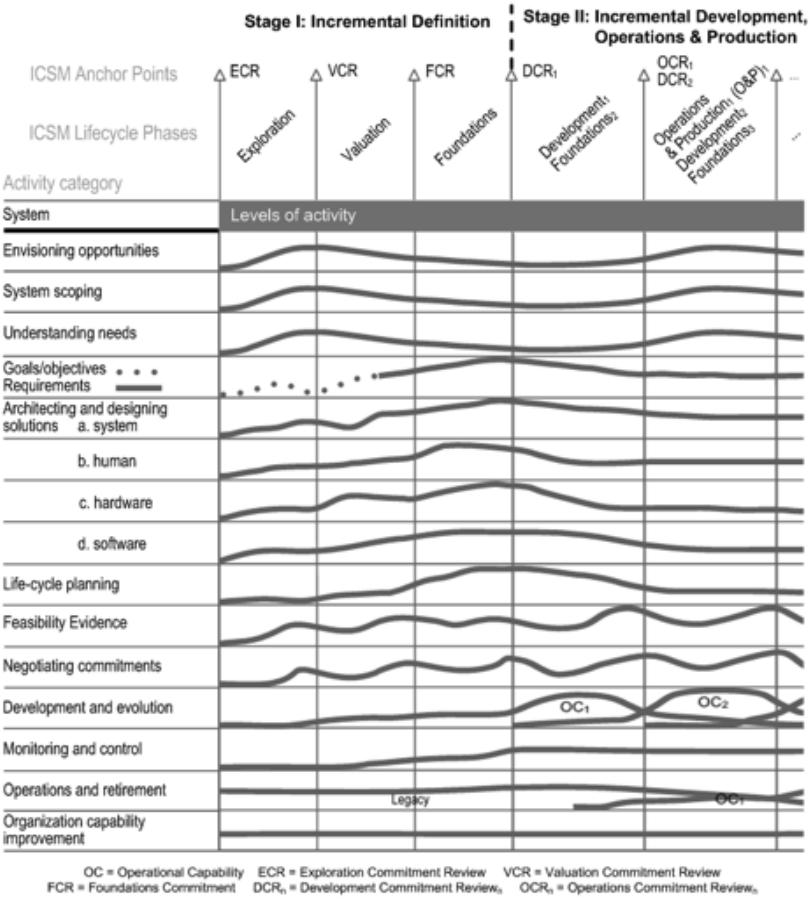

Figure 8. ICSM Activity Categories and Level of Effort (Pew and Mavor 2007). Reprinted with permission by the National Academy of Sciences, Courtesy of National Academies Press, Washington, D.C. All other rights are reserved by the copyright owner.

As with the RUP version, the magnitude and shape of the levels of effort will be risk-driven and likely to vary from project to project. Figure 8 indicates that a great deal of concurrent activity occurs within and across the various ICSM phases, all of which need to be "*synchronized and stabilized,*" a best-practice phrase taken from *Microsoft Secrets* (Cusumano and Selby 1996) to keep the project under control.

The review processes and use of independent experts are based on the highly successful AT&T Architecture Review Board procedures described in “Architecture Reviews: Practice and Experience” (Maranzano et al. 2005). Figure 9 shows the content of the feasibility evidence description. Showing feasibility of the concurrently developed elements helps synchronize and stabilize the concurrent activities.

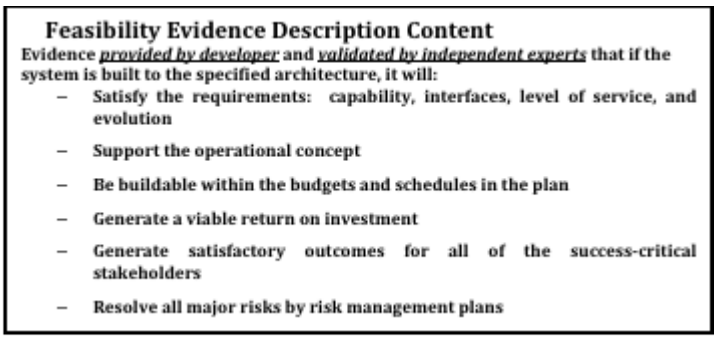

Figure 9. Feasibility Evidence Description Content (Pew and Mavor 2007).

The operations commitment review (OCR) is different in that it addresses the often-higher operational risks of fielding an inadequate system. In general, stakeholders will experience a two- to ten-fold increase in commitment level while going through the sequence of engineering certification review (ECR) to design certification review (DCR) milestones, but the increase in going from DCR to OCR can be much higher. These commitment levels are based on typical cost profiles across the various stages of the acquisition life cycle.

Underlying ICSM Principles

ICSM has four underlying principles which must be followed:

1. Stakeholder value-based system definition and evolution;
2. Incremental commitment and accountability;
3. Concurrent system and software definition and development; and
4. Evidence and risk-based decision making.

Model Experience to Date

The National Research Council Human-Systems Integration study (2008) found that the ICSM processes and principles correspond well with best commercial practices, as described in the Next Generation Medical Infusion Pump Case Study in Part 7. Further examples are found in *Human-System Integration in the System Development Process: A New Look* (Pew and Mavor 2007, chap. 5), *Software Project Management* (Royce 1998, Appendix D), and the annual series of "Top Five

Quality Software Projects", published in CrossTalk (2002-2005).

Advise this information below be deleted or moved to Part 6 because some of it will be redundant to the new article Agile Systems Engineering and the Lean part if redundant to the Lean Engineering article. These moves will drive updates/clean-ups needed to the references. ---

Agile and Lean Processes

According to the INCOSE *Systems Engineering Handbook* 3.2.2, *"Project execution methods can be described on a continuum from 'adaptive' to 'predictive.' Agile methods exist on the 'adaptive' side of this continuum, which is not the same as saying that agile methods are 'unplanned' or 'undisciplined,'"* (INCOSE 2011, 179). Agile development methods can be used to support iterative life cycle models, allowing flexibility over a linear process that better aligns with the planned life cycle for a system. They primarily emphasize the development and use of tacit interpersonal knowledge as compared to explicit documented knowledge, as evidenced in the four value propositions in the **"Agile Manifesto"**:

> *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; and*
> - ***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.* (Agile Alliance 2001)

Lean processes are often associated with agile methods,

although they are more scalable and applicable to high-assurance systems. Below, some specific agile methods are presented, and the evolution and content of lean methods is discussed. Please see "Primary References", "Additional References", and the Lean Engineering article for more detail on specific agile and lean processes.

Scrum

Figure 10 shows an example of Scrum as an agile process flow. As with most other agile methods, Scrum uses the evolutionary sequential process shown in Table 1 (above) and described in Fixed-Requirements and Evolutionary Development Processes section in which systems capabilities are developed in short periods, usually around 30 days. The project then re-prioritizes its backlog of desired features and determines how many features the team (usually 10 people or less) can develop in the next 30 days.

Figure 10 also shows that once the features to be developed for the current Scrum have been expanded (usually in the form of informal stories) and allocated to the team members, the team establishes a daily rhythm of starting with a short meeting at which each team member presents a roughly one-minute summary describing progress since the last Scrum meeting, potential obstacles, and plans for the upcoming day.

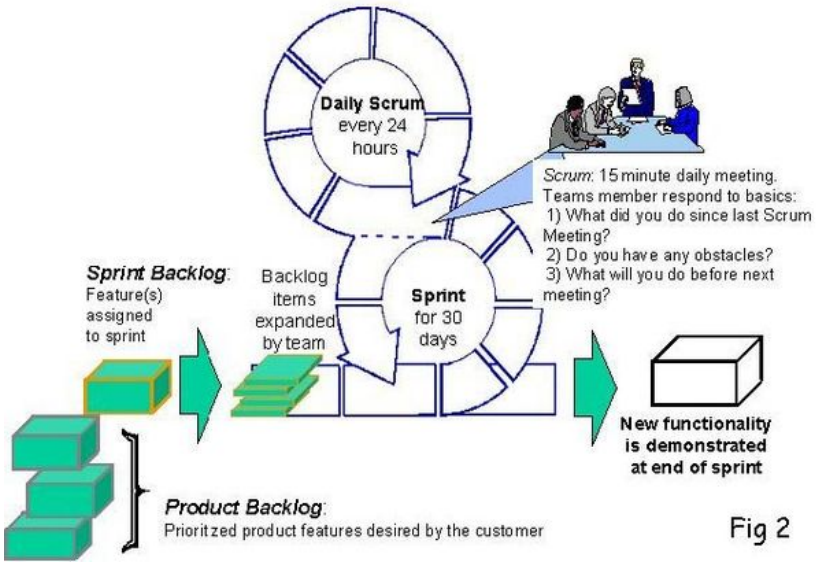

Figure 10. Example Agile Process Flow: Scrum (Boehm and Turner 2004). Reprinted with permission of Ken Schwaber. All other rights are reserved by the copyright owner.

Architected Agile Methods

Over the last decade, several organizations have been able to scale up agile methods by using two layers of ten-person Scrum teams. This involves, among other things, having each Scrum team's daily meeting followed up by a daily meeting of the Scrum team leaders discussing up-front investments in evolving system architecture (Boehm et al. 2010). Figure 11 shows an example of the Architected Agile approach.

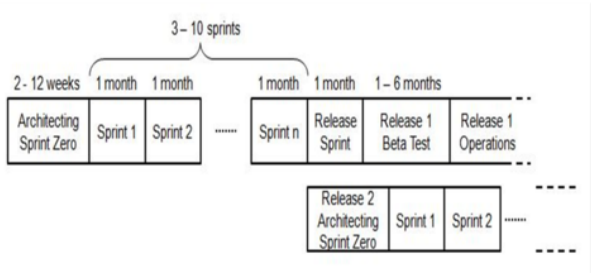

Figure 11. Example of Architected Agile Process (Boehm 2009). Reprinted with permission of Barry Boehm on behalf of USC-CSSE. All other rights are reserved by the copyright owner.

Agile Practices and Principles

As seen with the Scrum and architected agile methods, "generally-shared" principles are not necessarily "uniformly followed". However, there are some general practices and principles shared by most agile methods:

- The project team understands, respects, works, and behaves within a defined SE process;
- The project is executed as fast as possible with minimum down time or staff diversion during the project and the critical path is managed;
- All key players are physically or electronically collocated, and "notebooks" are considered team property available to all;
- Baseline management and change control are achieved by formal, oral agreements based on "make a promise—keep a promise" discipline. Participants hold each other accountable;
- Opportunity exploration and risk reduction are accomplished by expert consultation and rapid model verification coupled with close customer collaboration;
- Software development is done in a rapid development environment while hardware is developed in a multi-

disciplined model shop; and

- A culture of constructive confrontation pervades the project organization. The team takes ownership for success; it is never “someone else’s responsibility.”

Agile development principles (adapted for SE) are as follows (adapted from *Principles behind the Agile Manifesto* (Beedle et al. 2009)):

1. First, satisfy the customer through early and continuous delivery of valuable software (and other system elements).
2. Welcome changing requirements, even late in development; agile processes harness change for the customer’s competitive advantage.
3. Deliver working software (and other system elements) frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business personnel and developers must work together daily throughout the project.
5. Build projects around motivated individuals; give them the environment, support their needs, and trust them to get the job done.
6. The most efficient and effective method of conveying information is face-to-face conversation.
7. Working software (and other system elements) is the primary measure of progress.
8. Agile processes promote sustainable development; the sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.

A team should reflect on how to become more effective at regular intervals and then tune and adjust its behavior accordingly. This self-reflection is a critical aspect for projects that implement agile processes.

Lean Systems Engineering and Development

Origins

As the manufacturing of consumer products such as automobiles became more diversified, traditional pre-planned mass-production approaches had increasing problems with quality and adaptability. Lean manufacturing systems such as the Toyota Production System (TPS) (Ohno 1988) were much better suited to accommodate diversity, to improve quality, and to support just-in-time manufacturing that could rapidly adapt to changing demand patterns without having to carry large, expensive inventories.

Much of this transformation was stimulated by the work of W. Edwards Deming, whose Total Quality Management (TQM) approach shifted responsibility for quality and productivity from planners and inspectors to the production workers who were closer to the real processes (Deming 1982). Deming's approach involved everyone in the manufacturing organization in seeking continuous process improvement, or "Kaizen".

Some of the TQM techniques, such as statistical process control and repeatability, were more suited to repetitive manufacturing processes than to knowledge work such as systems engineering (SE) and software engineering (SwE). Others, such as early error elimination, waste elimination, workflow stabilization, and Kaizen, were equally applicable to knowledge work. Led by Watts Humphrey, TQM became the focus for the Software Capability Maturity Model (Humphrey 1987; Paulk et al. 1994) and the CMM-Integrated or CMMI, which extended its scope to include systems engineering (Chrissis et al. 2003). One significant change was the redefinition of Maturity Level 2 from "Repeatable" to "Managed".

The Massachusetts Institute of Technology (MIT) conducted studies of the TPS, which produced a similar approach that was called the "Lean Production System" (Krafcik 1988; Womack et al. 1990). Subsequent development of "lean thinking" and related work at MIT led to the Air Force-sponsored Lean Aerospace Initiative (now called the Lean Advancement Initiative), which applied lean thinking to SE (Murman 2003, Womack-Jones 2003). Concurrently, lean ideas were used to strengthen the scalability and dependability aspects of agile methods for software (Poppendieck 2003; Larman-Vodde 2009). The Kanban flow-oriented approach has been successfully applied to software development (Anderson 2010).

Principles

Each of these efforts has developed a similar but different set of Lean principles. For systems engineering, the current best source is *Lean for Systems Engineering*, the product of several years' work by the INCOSE Lean SE working group (Oppenheim 2011). It is organized into six principles, each of which is elaborated into a set of lean enabler and sub-enabler patterns for satisfying the principle:

1. **Value.** Guide the project by determining the value propositions of the customers and other key stakeholders. Keep them involved and manage changes in their value propositions.
2. **Map the Value Stream (Plan the Program).** This includes thorough requirements specification, the concurrent exploration of trade spaces among the value propositions, COTS evaluation, and technology maturity assessment, resulting in a full project plan and set of requirements.
3. **Flow.** Focus on the project's critical path activities to avoid expensive work stoppages, including coordination with external suppliers.
4. **Pull.** Pull the next tasks to be done based on prioritized needs and dependencies. If a need for the task can't be found, reject it as waste.
5. **Perfection.** Apply continuous process improvement to approach perfection. Drive defects out early to get the system Right The First #**Time,** vs. fixing them during inspection and test. Find and fix root causes rather than symptoms.
6. **Respect for People.** Flow down responsibility, authority, and accountability to all personnel. Nurture a learning environment. Treat people as the organization's most valued assets. (Oppenheim 2011)

These lean SE principles are highly similar to the four underlying incremental commitment spiral model principles.

- **Principle 1: Stakeholder value-based system definition and evolution**, addresses the lean SE principles of value, value stream mapping, and respect for people (developers are success-critical stakeholders in the ICSM).
- **Principle 2: Incremental commitment and**

accountability, partly addresses the pull principle, and also addresses respect for people (who are accountable for their commitments).

- **Principle 3: Concurrent system and software definition and development**, partly addresses both value stream mapping and flow.
- **Principle 4: Evidence and risk-based decision making**, uses evidence of achievability as its measure of success. Overall, the ICSM principles are somewhat light on continuous process improvement, and the lean SE principles are somewhat insensitive to requirements emergence in advocating a full pre-specified project plan and set of requirements.

See Lean Engineering for more information.

References

Works Cited

Agile Alliance. 2001. “Manifesto for Agile Software Development.” http://agilemanifesto.org/.

Anderson, D. 2010. *Kanban*, Sequim, WA: Blue Hole Press.

Boehm, B. 1996. "Anchoring the Software Process." IEEE *Software* 13(4): 73-82.

Boehm, B. and J. Lane. 2007. “Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering.” *CrossTalk*. 20(10) (October 2007): 4-9.

Boehm, B., J. Lane, S. Koolmanjwong, and R. Turner. 2010. “Architected Agile Solutions for Software-Reliant Systems,” in Dingsoyr, T., T. Dyba., and N. Moe (eds.), *Agile Software Development: Current Research and Future Directions.* New York, NY, USA: Springer.

Boehm, B. and R. Turner. 2004. *Balancing Agility and Discipline*. New York, NY, USA: Addison-Wesley.

Castellano, D.R. 2004. “Top Five Quality Software Projects.” *CrossTalk.* 17(7) (July 2004): 4-19. Available at: http://www.crosstalkonline.org/storage/issue-archives/2004/200407/200407-0-Issue.pdf.

Chrissis, M., M. Konrad, and S. Shrum. 2003. *CMMI:*

Guidelines for Process Integration and Product Improvement. New York, NY, USA, Addison Wesley.

Deming, W.E. 1982. *Out of the Crisis.* Cambridge, MA, USA: MIT.

Fairley, R. 2009. *Managing and Leading Software Projects.* New York, NY, USA: John Wiley & Sons.

Forsberg, K. 1995. "If I Could Do That, Then I Could…' System Engineering in a Research and Development Environment." Proceedings of the Fifth International Council on Systems Engineering (INCOSE) International Symposium. 22-26 July 1995. St Louis, MO, USA.

Forsberg, K., H. Mooz, and H. Cotterman. 2005. *Visualizing Project Management,* 3rd ed. New York, NY, USA: John Wiley & Sons.

Humphrey, W., 1987. "Characterizing the Software Process: A Maturity Framework." Pittsburgh, PA, USA: CMU Software Engineering Institute. CMU/SEI-87-TR-11.

Jarzombek, J. 2003. "Top Five Quality Software Projects." *CrossTalk.* 16(7) (July 2003): 4-19. Available at: http://www.crosstalkonline.org/storage/issue-archives/2003/200307/200307-0-Issue.pdf.

Krafcik, J. 1988. "Triumph of the lean production system". *Sloan Management Review.* 30(1): 41–52.

Kruchten, P. 1999. *The Rational Unified Process*. New York, NY, USA: Addison Wesley.

Larman , C. and B. Vodde. 2009. *Scaling Lean and Agile Development.* New York, NY, USA: Addison Wesley.

Maranzano, J.F., S.A. Rozsypal, G.H. Zimmerman, G.W. Warnken, P.E. Wirth, D.M. Weiss. 2005. "Architecture Reviews: Practice and Experience." IEEE *Software*. 22(2): 34-43.

Murman, E. 2003. *Lean Systems Engineering I, II, Lecture Notes*, MIT Course 16.885J, Fall. Cambridge, MA, USA: MIT.

Oppenheim, B. 2011. *Lean for Systems Engineering*. Hoboken, NJ: Wiley.

Paulk, M., C. Weber, B. Curtis, and M. Chrissis. 1994. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Reading, MA, USA: Addison

Wesley.

Pew, R. and A. Mavor (eds.). 2007. *Human-System Integration in The System Development Process: A New Look*. Washington, DC, USA: The National Academies Press.

Poppendieck, M. and T. Poppendieck. 2003. *Lean Software Development: An Agile Toolkit for Software Development Managers.* New York, NY, USA: Addison Wesley.

Spruill, N. 2002. "Top Five Quality Software Projects." *CrossTalk*. 15(1) (January 2002): 4-19. Available at: http://www.crosstalkonline.org/storage/issue-archives/2002/200201/200201-0-Issue.pdf.

Stauder, T. "Top Five Department of Defense Program Awards." *CrossTalk*. 18(9) (September 2005): 4-13. Available at http://www.crosstalkonline.org/storage/issue-archives/2005/200509/200509-0-Issue.pdf.

Womack, J., D. Jones, and D Roos. 1990. *The Machine That Changed the World: The Story of Lean Production*. New York, NY, USA: Rawson Associates.

Womack, J. and D. Jones. 2003. *Lean Thinking*. New York, NY, USA: The Free Press.

Primary References

Beedle, M., et al. 2009. "The Agile Manifesto: Principles behind the Agile Manifesto". in *The Agile Manifesto* [database online]. Accessed 2010. Available at: www.agilemanifesto.org/principles.html.

Boehm, B. and R. Turner. 2004. *Balancing Agility and Discipline*. New York, NY, USA: Addison-Wesley.

Fairley, R. 2009. *Managing and Leading Software Projects.* New York, NY, USA: J. Wiley & Sons.

Forsberg, K., H. Mooz, and H. Cotterman. 2005. *Visualizing Project Management,* 3rd ed. New York, NY, USA: J. Wiley & Sons.

INCOSE. 2012. *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. Version 3.2.2. San Diego, CA, USA: International Council on Systems Engineering (INCOSE), INCOSE-TP-2003-002-03.2.2.

Lawson, H. 2010. *A Journey Through the Systems Landscape.* Kings College, UK: College Publications.

Pew, R., and A. Mavor (eds.). 2007. *Human-System Integration in the System Development Process: A New Look*. Washington, DC, USA: The National Academies Press.

Royce, W.E. 1998. *Software Project Management: A Unified Framework*. New York, NY, USA: Addison Wesley.

Additional References

Anderson, D. 2010. *Kanban*. Sequim, WA, USA: Blue Hole Press.

Baldwin, C. and K. Clark. 2000. *Design Rules: The Power of Modularity.* Cambridge, MA, USA: MIT Press.

Beck, K. 1999. *Extreme Programming Explained.* New York, NY, USA: Addison Wesley.

Beedle, M., et al. 2009. "The Agile Manifesto: Principles behind the Agile Manifesto" in The Agile Manifesto [database online]. Accessed 2010. Available at: www.agilemanifesto.org/principles.html

Biffl, S., A. Aurum, B. Boehm, H. Erdogmus, and P. Gruenbacher (eds.). 2005. *Value-Based Software Engineering*. New York, NY, USA: Springer.

Boehm, B. 1988. "A Spiral Model of Software Development." IEEE *Computer.* 21(5): 61-72.

Boehm, B. 2006. "Some Future Trends and Implications for Systems and Software Engineering Processes." *Systems Engineering.* 9(1): 1-19.

Boehm, B., A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy. 1998. "Using the WinWin Spiral Model: A Case Study." IEEE *Computer.* 31(7): 33-44.

Boehm, B., J. Lane, S. Koolmanojwong, and R. Turner. 2013 (in press). *Embracing the Spiral Model: Creating Successful Systems with the Incremental Commitment Spiral Model.* New York, NY, USA: Addison Wesley.

Castellano, D.R. 2004. "Top Five Quality Software Projects." *CrossTalk.* 17(7) (July 2004): 4-19. Available at: http://www.crosstalkonline.org/storage/issue-archives/2004/200407/200407-0-Issue.pdf.

Checkland, P. 1981. *Systems Thinking, Systems Practice*. New York, NY, USA: Wiley.

Crosson, S. and B. Boehm. 2009. “Adjusting Software Life Cycle Anchorpoints: Lessons Learned in a System of Systems Context.” Proceedings of the Systems and Software Technology Conference, 20-23 April 2009, Salt Lake City, UT, USA.

Dingsoyr, T., T. Dyba. and N. Moe (eds.). 2010. "Agile Software Development: Current Research and Future Directions.” Chapter in B. Boehm, J. Lane, S. Koolmanjwong, and R. Turner, *Architected Agile Solutions for Software-Reliant Systems.* New York, NY, USA: Springer.

Dorner, D. 1996. *The Logic of Failure*. New York, NY, USA: Basic Books.

Faisandier, A. 2012. *Systems Architecture and Design*. Belberaud, France: Sinergy'Com.

Forsberg, K. 1995. "'If I Could Do That, Then I Could…' System Engineering in a Research and Development Environment.” Proceedings of the Fifth Annual International Council on Systems Engineering (INCOSE) International Symposium. 22-26 July 1995. St. Louis, MO, USA.

Forsberg, K. 2010. “Projects Don’t Begin With Requirements.” Proceedings of the IEEE Systems Conference. 5-8 April 2010. San Diego, CA, USA.

Gilb, T. 2005. *Competitive Engineering*. Maryland Heights, MO, USA: Elsevier Butterworth Heinemann.

Goldratt, E. 1984. *The Goal*. Great Barrington, MA, USA: North River Press.

Hitchins, D. 2007. *Systems Engineering: A 21st Century Systems Methodology.* New York, NY, USA: Wiley.

Holland, J. 1998. *Emergence*. New York, NY, USA: Perseus Books.

ISO/IEC. 2010. *Systems and Software Engineering, Part 1: Guide for Life Cycle Management.* Geneva, Switzerland: International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC), ISO/IEC 24748-1:2010.

ISO/IEC/IEEE. 2015. *Systems and Software Engineering -- System Life Cycle Processes.* Geneva, Switzerland: International Organisation for Standardisation /

International Electrotechnical Commissions. ISO/IEC/IEEE 15288:2015.

ISO/IEC. 2003. *Systems Engineering — A Guide for The Application of ISO/IEC 15288 System Life Cycle Processes.* Geneva, Switzerland: International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC), ISO/IEC 19760:2003 (E).

Jarzombek, J. 2003. "Top Five Quality Software Projects." *CrossTalk.* 16(7) (July 2003): 4-19. Available at: http://www.crosstalkonline.org/storage/issue-archives/2003/200307/200307-0-Issue.pdf.

Kruchten, P. 1999. *The Rational Unified Process.* New York, NY, USA: Addison Wesley.

Landis, T. R. 2010. *Lockheed Blackbird Family (A-12, YF-12, D-21/M-21 & SR-71).* North Branch, MN, USA: Specialty Press.

Madachy, R. 2008. *Software Process Dynamics*. New York, NY, USA: Wiley.

Maranzano, J., et al. 2005. "Architecture Reviews: Practice and Experience." IEEE *Software.* 22(2): 34-43.

National Research Council of the National Academies (USA). 2008. *Pre-Milestone A and Early-Phase Systems Engineering*. Washington, DC, USA: The National Academies Press.

Osterweil, L. 1987. "Software Processes are Software Too." Proceedings of the SEFM 2011: 9th International Conference on Software Engineering. Monterey, CA, USA.

Poppendeick, M. and T. Poppendeick. 2003. *Lean Software Development: an Agile Toolkit.* New York, NY, USA: Addison Wesley.

Rechtin, E. 1991. *System Architecting: Creating and Building Complex Systems.* Upper Saddle River, NY, USA: Prentice-Hall.

Rechtin, E., and M. Maier. 1997. *The Art of System Architecting*. Boca Raton, FL, USA: CRC Press.

Schwaber, K. and M. Beedle. 2002. *Agile Software Development with Scrum*. Upper Saddle River, NY, USA: Prentice Hall.

Spruill, N. 2002. “Top Five Quality Software Projects.” *CrossTalk*. 15(1) (January 2002): 4-19. Available at: http://www.crosstalkonline.org/storage/issue-archives/2002/200201/200201-0-Issue.pdf.

Stauder, T. 2005. “Top Five Department of Defense Program Awards.” *CrossTalk*. 18(9) (September 2005): 4-13. Available at http://www.crosstalkonline.org/storage/issue-archives/2005/200509/200509-0-Issue.pdf.

Warfield, J. 1976. *Societal Systems: Planning, Policy, and Complexity*. New York, NY, USA: Wiley.

Womack, J. and D. Jones. 1996. *Lean Thinking*. New York, NY, USA: Simon and Schuster.

< Previous Article | Parent Article | Next Article >
SEBoK v. 2.10, released 06 May 2024
