# In Search of a Modern Software Life Cycle

# Secure DevOps Foundations for Large-Scale Software Systems

**Don O'Neill**

**Abstract.** "In Search of a Modern Software Life Cycle" explores the "Secure DevOps Foundations for Large Scale Software Systems" in terms of voices from the trenches, the field of play, life cycle on center stage, and evolutionary features and issues including sequential, prototype, incremental, iterative, spiral, CMMI©, technical debt, code and upload and frequency of release, next generation software engineering, open source software, false claims, integration engineering, and a new way of thinking.

## Heard from the Trenches

If DevOps is needed to change the world, Secure DevOps is also needed to save the world. In a world where business questions masquerade as technical questions, where programmers must experience an epiphany before they are motivated to master the skill of writing secure code, [1] and where bonuses must be withheld to obtain management attention to security, resistance rules.

If these are the risks, what are the outcomes? Acquirers complain they don't know how to ask for secure code from vendors, adding that they get what they ask for but not what they want. [2] It's complicated! Programmers confess that writing code is hard, and writing secure code may be beyond the tipping point. [3] Software engineers wonder if there is any secure code anywhere and assert that best practices are insufficient. Supply Chain Risk Management Software Assurance practitioners retreat behind the wall and only hope for bug-free, patchable software deliveries accompanied by a bill of material. [4] These were just some of the comments made at the 2016 CERT Secure Coding Symposium conducted by the Software Engineering Institute in Arlington, Virginia, on September 8, 2016.

## The Field of Play

Formed to support the advancement of software engineering in the Department of Defense (DoD), the Software Engineering Institute (SEI) lost its way by too vigorously pursuing commercial partners. Like the dog that chased and caught the firetruck without a plan for what comes next, the SEI lost its DoD sponsor, its principal foundation of financial support.

The impact of this lost sponsorship was most keenly felt by the Capability Maturity Model Integration (CMMI©) program, once the crown jewel of the SEI and Carnegie Mellon University (CMU) itself. Forced to depart the protection of the SEI and CMU, the CMMI© has now landed at the Information Systems Audit and Control Association (ISACA) in the form of the CMMI© Institute, relegated to serving the commercial IT governance professionals it catered to. Finding itself now in the competitive death grip of a more innovative and popular Agile method, the CMMI© framework continues to teeter. All this is occurring despite the fact that the value of the CMMI© has not yet been fully discovered (CrossTalk, 2012) despite a quarter-century of use. Yet there may still exist a way forward in harmonizing Agile and CMMI© (CrossTalk, 2016) as part of that discovery.

Even beyond the CMMI©, the broader software situation is dire (Defense AT&L, 2015). Industry and government continue to increase dependence on software produced by an immature profession that has stumbled in delivering trustworthy software components, systems, and systems of systems to the nation's critical infrastructure and defense industrial base. The result is cybersecurity weaknesses and vulnerabilities exploited at will by persistent adversaries whose capabilities and motivation can only be surmised by assessing their consequences.

## Center Stage

At play on center stage in all this is the software development life cycle. Beginning with Winston Royce, managing the development of large software systems became the center of attention based on a waterfall model of software activities and his belated inclusion of prototyping as an essential step (Royce, 1970).

From Royce's waterfall life cycle model followed by incremental, iterative, and spiral to the SEI's CMMI© followed by Agile methods and now DevOps, the software development life cycle continues as an unsettled issue. Today's unbridled complexity (Sheard, 2015), the stresses of scale in the Internet of Things (IoT) (Recode, 2016) with its explosion of endpoints and no one in charge, and the unpredictability of cybersecurity threats (CrossTalk, 2011) with their persistence of vulnerabilities like System 7 and its public safety access points all combine to destabilize software system development life cycle approaches.

At any point in time, Secure DevOps processes must possess the capability to detect cyber vulnerabilities and malware. Common Weaknesses Evaluation (CWE) and Common Vulnerabilities Evaluation (CVE) assist in this, as do tools like Hyperion from Oak Ridge, Function Extraction (FX) from CMU, MUSE from the Defense Advanced Research Projects Agency (DARPA), and Approximate Matching from the National Institute of Standards and Technology (NIST). Beyond the range of typical Secure DevOps, the sectors of the critical infrastructure with their stovepipe yet interdependent operations face more insidious supply chain resilience challenges (CrossTalk, 2014). And then there are cascade triggers. Hidden or in plain sight, cascade triggers are capable of invading various industry sectors in a variety of ways:

- The transportation sector can be brought to its knees if truck drivers cannot use credit cards to charge for gas tank fill-ups.
- The medical sector depends on the Internet to distribute and present patient electronic medical records.
- The electrical grid depends on a survivable electrical grid with predictable demand profiles matched to planned resources and capacities (Koppel, 2015).
- The banking and finance sector remains ever conscious of its need to protect next-day opening, even in the presence of a flash crash disruption (Lewis, 2014).
- The users of the telecommunications sector are increasingly vulnerable to Internet disruptions like Distributed Denial of Service (DDoS) and encryption-based scams like ransomware.

## Evolutionary Features and Issues

The following life cycle evolutionary features and consequences are introduced, including sequential, prototype, incremental, iterative, spiral, CMMI©, technical debt, code and upload and frequency of release, next generation software engineering, open source software, false claims, integration engineering, and a new way of thinking.

## Sequential

The much-aligned waterfall model is a linear sequence of dependent activities. Much of the focus on life cycle model improvement is devoted to disrupting this dependence on the sequential.

## Prototype

The use of prototypes — perhaps rapid prototypes — is an attempt to produce an early kernel of operational capability that can be exercised (not so much tested) to glean necessary insights into selective component interactions, numerical analysis of algorithms and their finite word effects, computer capacity utilization of both memory and speed, and targeted operational usage considerations.

## Incremental

The use of multi-level design (Defense AT&L, 2012) and staged incremental development (SSJ, 1983) are tactics to put early performance pressure on the development team and its people, processes, and tools through incremental stages of production; for example, operating system services, middleware, and environment; executing system and subsystem interfaces using underlying stubs; executing prime mode functionality buildup in place of stubs; and exercising and transitioning degraded mode scenarios.

## Iterative

Larman skillfully traces the real-world application of various evolutionary features in his "Agile & Iterative Development: A Manager's Guide" (Larman, 2004). Larman mentions the work of the IBM Federal Systems Division (FSD) on the integration engineering of the Trident Submarine Command and Control System (SSJ, 1983) and its pioneering work on design, development, and management life cycle activities spanning advanced design, systematic design, systematic programming, code management, integration engineering, technical reviews, cost management, and program management (IBM SJ, 1980).

## Spiral

Introduced by Barry Boehm, the foundational spiral method is a purposeful and strategic departure from the sequential waterfall model in integrating prototype, incremental, and iterative tactics in the systematic management of software system risk (Boehm, 2015).

## CMMI©

Now that the CMMI© has been organized into three constellations for assuring an organization's capability to perform development, acquisition, and service, there is a need to extend the range of value of the CMMI© to a new normal (CrossTalk, 2012). As an organization improves its process maturity, strategic imperatives need to replace waste and neglect as the CMMI© value driver. Only those organizations able to elevate their game and transition from tactical to strategic use of the CMMI© will be able to reap its full value.

While the traditional treatment of the value of the CMMI© in terms of cost, schedule, productivity, quality, customer satisfaction, and return on investment is sufficient to promote adoption of the CMMI© and even to sustain a process improvement initiative through the early maturity levels, the value of the CMMI© determined in this way is likely to be underestimated as the organization approaches higher maturity levels.

The value of the CMMI© can be framed more strategically as a means for carrying out visionary statements of strategic intent in achieving measured outcomes in business and competitiveness, management and predictability, process and improvement, engineering and trustworthiness, and operations and dependability.

© CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

## Technical Debt

Technical debt is the organizational, project, or engineering neglect of known good practices that can result in persistent public, user, customer, staff, reputation, or financial cost (Defense AT&L, 2013). Shortcuts, expedient activities, and poor practices that contribute to the initial product launch or initial operational capability are often cited as justifiable excuses for taking on technical debt. But in truth, most technical debt is taken on without this strategic intent, without even knowing it, and without the capacity to do the job right.

## Code and Upload and Frequency of Release

In order to simplify, relieve stress and sustain a very high frequency of release, one major corporation is employing an extreme move. They no longer test software upgrades, preferring instead to use the code-and-upload tactic. This leaves any defects to be encountered by unsuspecting customers. The frequency of release cited by this corporation is an amazing 30,000 per year.

## Next Generation Software Engineering

Practical Next Generation Software Engineering addresses the unclaimed benefits and unmet needs associated with competitiveness, security and software. In accordance with the austerity of the times, the immediate goal of practical next generation software engineering is to drive systems and software engineering to do “more with less … fast” (IEEE, 2009). Four practical objectives are identified to advance this goal using smart, trusted technologies:

- Drive user domain awareness.
- Simplify and produce systems and software using a shortened development life cycle.
- Compose and field trustworthy applications and systems from parts.
- Compose and operate resilient systems of systems from systems.

## Open Source Software

Open Source Software is openly available off-the-shelf software that depends on community development and distribution support subject to license compliance. Open source code is openly available for inspection and change. By contrast, closed source is a proprietary product dependent on the vendor for support and not open to inspection or modification.

Open source software features free distribution of source code. When open source software is extended or revised, the result is termed a “derived work.” Furthermore, an open source software license may permit resale of a derived work. While freely available, there are project costs associated with modifying and integrating derived works into deployable software systems.

The proper use, reuse, modification, and sale of open source software as derived work lies in the art of program and contract management. When this is done in government contracting, retaining the classification of “Commercial Off the Shelf” (COTS) and “Government Off the Shelf” (GOTS) software has financial and legal consequences. Furthermore, blending all this into use under the General Services Administration (GSA) contract may introduce complexities not yet fully explored. The government is recognizing the potential savings in absorbing software into the GOTS classification and is now establishing target goals for accomplishing this. Failure to assign the proper COTS and GOTS classifications and associated fee structures may result in a Department of Justice (DOJ) false claims charge against the contractor under the False Claims Act.

## False Claims

With 80 percent of government software procured as COTS and accorded limited or restricted rights, government acquisition managers need to be aware of intellectual property considerations (Defense AT&L, 2014). When modified and extended through government funding, COTS software becomes GOTS software and is entitled to government purpose rights. Unless the government acquisition manager insists on it, a contractor may engage in false claims practice by improperly marketing and selling GOTS software products as COTS. Instead of receiving the benefits of government purpose rights, the government may be charged a commercial product licensing fee and accorded only limited or restricted rights. Neglecting intellectual property rights can be costly.

## Integration Engineering

The penultimate challenge in fielding large-scale systems and systems of systems that are trustworthy, secure and resilient resides in critical infrastructure (White House, 2016). Simply put, the resilience value proposition is intended to yield a critical infrastructure capable of anticipating, avoiding, withstanding, minimizing, and recovering from the effects of adversity, whether natural or man-made, under all circumstances. This is based on an architecture of resilience that squarely faces the issues of harmonizing a diverse industry sector culture and context and offers effective prescriptions for success in the form of well-trained intelligent middlemen, a resiliency maturity framework, a system of systems technical architecture, a common and useful way of working, and an integration engineering program structure staffed by a capable resilience integrator. Anticipation and avoidance replace cleanup, recovery, and opportunity loss.

The author offers the following integration engineering context and culture harmonization guidance:

- Formality within an architectural framework facilitates the imposition of distributed supervisory control, interoperability, and operation sensing and monitoring protocols.
- Strong code management practices facilitate reconfiguration and reconstitution.
- Exercising strong control over the workforce facilitates business continuity and survivability.
- Exercising strong government control facilitates compliance for the benefit of the commons at the expense of initiative for the self-interest.
- The diverse industry sector expectations of trust, loyalty, and satisfaction must be respected, blended, and harmonized.
- Technical debt must be eliminated.
- Cascading and propagating triggers must be anticipated,

avoided, and minimized.
- Industry sector software sourcing exposures must be understood and managed.
- Supply chain risk management operations must be assured.
- Cybersecurity strategy policy decisions and defined tactics must be assured.

## A New Way of Thinking

The Integration Engineers, Resilience Integrators, and Intelligent Middlemen must be equipped with a new way of thinking. (Jacobson, I., Lawson, H.B., 2015). As the twig is bent, so grows the tree. To get your project off on the right foot, expectations should be set and evidence should be sought on the following assertions and principles:
- Stakeholders are in agreement and share a vision for the project.
- An opportunity value proposition has been established, and stakeholders share a vision for achieving it.
- Requirements or user stories are coherent and acceptable, and stakeholders share a vision for them.
- The software system architecture is selected and comprises a domain-specific architecture to guide software system implementation. The software system implementation is also made ready and operational with no technical debt.
- The team operates in collaboration, shares a vision for the project, and is ready to perform with respect to shared vision, software engineering processes, software project management, software product engineering, operations support, and domain-specific architecture processes, methods, and tools.
- The way of working by the team has established foundations for software engineering processes, software project management, software product engineering, and operations support.
- Work begins only when everything is prepared, including coherent requirements and acceptable user stories, stakeholders that are in agreement, and an established foundation for the way of working.
- All work products are prepared and inspected in accordance with a defined standard of excellence assuring completeness, correctness and consistency.

A product focus on perfection is assisted by the “work product” expectations as shown here. The work product should be:
- Identified as part of the way of working.
- Produced, shared with the team, and inspected.
- Complete with parts that are traceable to predecessor work products.
- Correct with parts that are verified and provably correct.
- Consistent in style and form of recording, and consistent with the software system architecture and its rules of construction.
- “Value add,” traceable to user stories and the “Done” criteria for the way of working.

## Conclusion

Clearly the search for an ideal model software life cycle is a journey, not a destination. The disruptive journey continues, with the tension of Agile and cybersecurity serving as current disrupters. As before, a variety of adaptations and innovations will emerge from practice, and some will be absorbed in the body of professional practice for those that follow. And so goes the evolution of the software profession.

## ABOUT THE AUTHOR

**Don O’Neill** served as the president of the Center for National Software Studies (CNSS) from 2005 to 2008. Following 27 years with IBM’s Federal Systems Division (FSD), he completed a three-year residency at Carnegie Mellon University’s Software Engineering Institute (SEI) under IBM’s Technical Academic Career Program and has served as an SEI visiting scientist. A seasoned software engineering manager, technologist, independent consultant, and expert witness, he has a Bachelor of Science degree in mathematics from Dickinson College in Carlisle, Penn. His current research is directed at public policy strategies for deploying resiliency in the nation’s critical infrastructure; disruptive game-changing fixed price contracting tactics to achieve DOD austerity; smart and trusted tactics and practices in supply chain risk management assurance; a defined “software clean room method” for transforming a proprietary system into a clean system devoid of proprietary information, copyrighted material, and trade secrets and confirming, verifying, and validating the results; and a constructive approach to sequencing the transition of SEMAT Essence Kernel Alpha states with an eye to pinpointing the risk triggers that threaten success and lead to the accumulation of technical debt.

## NOTES

1. David Svoboda. (Sept. 8, 2016.) SEI CERT Secure Coding Team, 2016 CERT Secure Coding Symposium. Arlington, Virginia.
2. Kris Britton. (Sept. 8, 2016.) NSA Center for Assured Software, 2016 CERT Secure Coding Symposium. Arlington, Virginia.
3. Dr. Carl Woody. (Sept. 8, 2016.) CERT Cyber Security Engineering Team, 2016 CERT Secure Coding Symposium. Arlington, Virginia.
4. Josh Corman. (Sept. 8, 2016.) The Atlantic Council, 2016 CERT Secure Coding Symposium. Arlington, Virginia.

## REFERENCES

O’Neill, D. (January/February 2012.) “Extending the Value of the CMMI to a New Normal.” CrossTalk, The Journal of Defense Software Engineering. http://www.crosstalkonline.org/storage/issue-archives/2012/201201/201201-ONeill.pdf.

O’Neill, D. (July/August 2016.) “The Way Forward: A Strategy for Harmonizing Agile and CMMI.” CrossTalk, The Journal of Defense Software Engineering. http://static1.1.sqspcdn.com/static/f/702523/27124563/1466890559753/201607-ONeill.pdf?token=CYClitqj%2B4Q5KzD%2B8d1nHTuHh9s%3D.

## REFERENCES CONT.

O'Neill, D. (May/June 2015.) "Software 2015: Situation Dire." Defense Advanced Technology and Logistics (DAT&L) Magazine. http://www.dau.mil/publications/DefenseATL/DATLFiles/May-Jun2015/O'Neill.pdf.

Royce, Winston W. (August 25–28, 1970.) "Managing the Development of Large Software Systems." Technical Papers of Western Electronic Show and Convention (WesCon). Los Angeles, Calif., USA.

Sheard, Sarah. (2015.) "Chapter 5: Complexity, Systems, and Software, 'Software Engineering in the Systems Context.'" Edited by Ivar Jacobson and Harold "Bud" Lawson. College Publications, King's College, London. ISBN 978-1-84890-76-6. 578 pages;

O'Neill, D. (1983.) "Integration Engineering Perspective." The Journal of Systems and Software, 3. 77-83. http://www.sciencedirect.com/science/article/pii/0164121283900067.

O'Donnell, Bob. (June 22, 2016.) "The Internet of Things is facing challenges with scale." Recode. http://www.recode.net/2016/6/22/11991414/internet-of-things-iot-challenges-scale.

O'Neill, D. (September/October 2011.) "Cyber Strategy, Analytics, and Tradeoffs: A Cyber Tactics Study." CrossTalk, The Journal of Defense Software Engineering. http://www.crosstalkonline.org/storage/issue-archives/2011/201109/201109-ONeill.pdf.

O'Neill, D. (March/April 2014.),"Software and Supply Chain Risk Management Assurance Framework." CrossTalk, The Journal of Defense Software Engineering. http://www.crosstalkonline.org/storage/issue-archives/2014/201403/201403-ONeill.pdf.

Koppel, T. (2015.) "Lights Out." Crown Publishing Group. ISBN 978-0-553-41996-2. 277 pages.

O'Neill, D. (1983.) "Integration Engineering Perspective." The Journal of Systems and Software, 3, 77-83. http://www.sciencedirect.com/science/article/pii/0164121283900067.

Larman, C. (2004.) "Agile & Iterative Development: A Manager's Guide." Pearson Education, Inc. ISBN 0-13-111155-8, 82-85.

O'Neill, D., Linger, R.C., Dyer, M. & Quinnan, R.E. (1980.) "The Management of Software Engineering." IBM Systems Journal, Vol. 19, Number 4, 414-477. http://www.research.ibm.com/journal/sj/.

Boehm, Barry. (2015.) "Chapter 6: Principles and Rationale for Successful Systems and Software Processes, 'Software Engineering in the Systems Context.'" Edited by Ivar Jacobson and Harold "Bud" Lawson. College Publications, King's College, London. ISBN 978-1-84890-76-6. 578 pages.

O'Neill, D. (March/April 2013.) "Technical Debt in the Code: Cost to Software Planning." Defense Advanced Technology and Logistics (DAT&L) Magazine. http://www.dau.mil/pubscats/ATL%20Docs/Mar_Apr_2013/O%27Neill.pdf.

O'Neill, D. (June 2009.) "Preparing the Ground for Next Generation Software Engineering." IEEE Reliability Society, Annual Technology Report 2008, 148-151.

O'Neill, D. (November/December 2014.) "Avoiding Proprietary Problems: A Software Clean-Room Method." Defense AT&L Magazine. http://www.dau.mil/publications/DefenseATL/DATLFiles/Nov-Dec2014/O'Neill.pdf.

O'Neill, D. (April 14, 2016.) "Integration Engineering in the Pursuit of Critical Infrastructure Resilience: A Unified Theory." White House Cyber Commission on Enhancing National Cybersecurity, Kickoff Meeting. http://www.nist.gov/cybercommission/upload/Meeting_Minutes_April_14.pdf.

Jacobson, I. & Lawson, H.B. (2015.) "Software Engineering in the Systems Context." Edited by Ivar Jacobson and Harold "Bud" Lawson. College Publications, King's College, London. ISBN 978-1-84890-76-6. 578 pages.

Lewis, Michael. (2014.) "Flash Boys: A Wall Street Revolt." W.W. Norton and Company, Ltd. ISBN 978-0-393-24466-3. 274 pages.

# The Software Deployment Process and Automation

**Dr. Nary Subramanian, Associate Professor of Computer Science, University of Texas at Tyler**

**Abstract.** Software deployment is the last step in the software development life cycle. During deployment, control of the software transfers from the development team to the customer. After deployment, people in the customer organization will use the software as part of their jobs and derive economic benefits from the software. Any defects found in software post-deployment are resolved as part of the maintenance phase. The first step in mitigating user problems is the proper deployment of software. Software deployment is anything but trivial. Some enterprise software may take months, if not years, to completely deploy. Therefore, efficient software deployment will considerably shorten the deployment phase and save resources in terms of cost and labor. In this article, we explore typical models for software deployment. Based on these models, we develop a generic software deployment model, then identify deployment processes that lend themselves to further automation and may lead to an overall reduction in the deployment effort.

## 1. Introduction

Software deployment or installation represents the final handover of software from the development team to the customer. After successful deployment, the software system is finally operational so that the customer can benefit economically from its use. At the end of this deployment effort, the software development organization receives payment from the customer and the project is considered successful from both the developer's and the customer's viewpoints. However, software deployment is anything but trivial, depending on the scale of implementation. While a nontechnical person can install a desktop application by either installing a downloaded file or installing from a disk, a large-scale enterprise resource planning (ERP) system such as SAP may take several months — if not years — to be fully configured and ready to use [1, 2, 3].

A question one might have is why certain software deployments take a long time. Is it possible to shorten all deployments to the time it takes to install a desktop application? In this article we examine typical deployment models and discuss some answers to these questions. To answer these questions, we develop a generic deployment model based on typical deployment models, and this generic model will help us rationalize our answers. We also explore opportunities to automate some or all deployment activities.

What happens when, after successful software deployment, users notice defects (or bugs) during normal software operation? The customer reports these bugs to the software develop-