



# Reaching Out to the Future

The goal of this chapter is to outline the mindset needed to survive and thrive as a software engineering professional in this digital age. Software engineering professionals must not only know technology, design, and how to build products, but also be able to contribute to their way of working, methods, and practices. They may not have the expertise to design their team's or organization's way of working, but they must participate in it because it affects them. In this chapter, the reader will be shown

- the difference between agility as “doing agile” (with practices) compared to “being agile” (with mindset) and explain why the latter is more important than the former;
- important mindset changes to be aware of, including team ownership (instead of selected-few ownership), a focus on method use rather than description, and embracing the evolution of the methods (instead of keeping them fixed);
- important shortcomings in method descriptions, e.g., failing to communicate what the team really needs—which might be a way to determine real development progress, a means to plan, a system to organize their team members, etc.; and
- the important role of continuous improvement and intentional creation of an environment, in which continuous improvement of the employed method can naturally happen.

The intent is to encourage readers to research and evolve a method that helps teams and organizations deliver software much better.

This is an exciting day and age in which to be software professionals. We are at the forefront of every transformation in every business and every nation. Businesses are becoming digital. Nations are becoming digital. There are rapid technological changes every day, with lowering costs of computing and bandwidth. We see

augmented and virtual reality applications becoming commonplace. Instead of traveling to a new place, you can do it through your headset. Instead of merely looking at a building design on paper, you can walk through and experience it using virtual or augmented reality before confirming your wishes to the architect. Electric cars are common now, and we are at the advent of large-scale proliferation of autonomous vehicles and a revolution in transportation. Artificial intelligence applications are also becoming popular, and they can do things that before we could only have imagined in science fiction. We could go on forever talking about these exciting advances, but we want to pause and ponder their implications for software engineering.

In the midst of all this, software is at the forefront and at the core of this change. It is not presumptuous to say that the world runs on software. With all these changes, some very natural questions software professionals can ask are: What will software engineering be like in the future? Will it be something revolutionary and brand new, requiring complete overhaul? Or will it be an evolution of what we already have established?

These were the same questions that plagued Smith, who had recently been promoted to become the head of software engineering for TravelEssence. He needed to build the competency of his teams and establish an effective way of working to face rapid advancement and strong competition. His teams were adopting new tools like advanced analytics, blockchain, AI, and many more, and they needed to collaborate effectively. These advances entailed different kinds of development, and Smith discovered that with all of them there would be something “old” and something “new.”

There would be something “old” in the sense that there are timeless principles and practices that were useful even with the newest advances. In particular, the kernel was always helpful in evaluating the progress and health of teams’ endeavors. Practices like Scrum Lite, Use-Case Lite, and Microservices Lite, presented in Chapters 15–17, and the program and team practices discussed earlier in this part of the book, were still applicable even for these new technologies.

Nevertheless, there would be something new, as expected. New applications, such as advanced analytics, required new competencies and new practices. For example, they required an Analytics Translator competency: someone able to take vague goals like “I want to increase sales by 50%” and translate them into actionable data requirements that could then be implemented by development teams. Even the recommendation engine that Smith and his team had implemented (see Part III) was already, in fact, an instance of an AI application, as that system could intelligently recommend and guide users in their bookings and travels.

Because Smith and TravelEssence had their methods and practices essentialized, they were able to add new practices easily. Team members knew how the new practices fit. This wouldn't have been the case if TravelEssence had been using monolithic methods. Their early investment into essentialization had paid off.

Breaking from our fictional TravelEssence story, in the larger scheme of things, the authors of this book and the SEMAT community have been reaching out to different kinds of development across the industry. An early result is the application of Essence to enterprise IoT. In simplistic terms, enterprise IoT systems comprise many smart devices (e.g., smart watches, wearables, light bulbs, sensors, etc.) connected to and sending information to powerful enterprise backends to perform sophisticated computations. Enterprise IoT powers smart buildings and even smart cities. What are software engineering practices like for these applications? Interested readers may refer to [Jacobson et al. 2017] for details. At the same time, the SEMAT community is also building a library of practices with participation from industry leaders. Methods such as Scrum and Disciplined Agile have already been essentialized and made available. All these efforts are designed to help software teams and organizations deliver better software better, faster, cheaper, and happier.

## 23.1 Be Agile with Practices and Methods

In this digital age, being agile with practices and methods is crucial. Even back in the late 1990s, software engineering experts had recognized this fact: in our fast-changing Darwinian world, survival is about the ability to adapt quickly. In 2001, a group of software engineering pragmatists came together to discuss how to overcome challenges faced by our industry. At that time, most development endeavors were using a so-called “waterfall approach,” which took years to complete, and they were not really delivering what customers or users needed. These experts had been successful with their (at that time) new approaches and were constantly improvising and innovating. They summarized what they believed in as a 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
- 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.”<sup>1</sup>

Perhaps the most agile statement in the manifesto is that “We are uncovering better ways of developing software by doing it and helping others do it.” It is this continuous striving to be better that spurs learning and advances in software engineering, not just in the industry, but in every team. Essence has taken the additional step to make the capturing, and communication—and hence the learning and application—of appropriate practices better. But this is not enough.

We recognize that every team has its own unique development context; they are engaged in different kinds of development, their members have different levels of experience and backgrounds. They can be working on small- or large-scale development. But in all these situations, a team can always collaborate and grow together to become better. It is this continuous adaptation to the development context and continuous learning that being agile with practices and methods is fundamentally about.

The authors of the Essence book [Jacobson et al. 2013a] take the values and principles of the agile movement to another level by providing tools for teams to explicitly evolve their way of working.

In contrast to many previous method initiatives, the focus of SEMAT is on those who know best what works and what doesn’t work when building software systems: that is, architects, analysts, designers, programmers, testers, and software managers. This takes a change in mindset—changes in the ways you think about methods—just as moving to agile development from traditional development (e.g., the waterfall lifecycle) requires a change in mindset on how to develop software.

What are the specific mindset changes you need to be aware of? To some extent they differ from the Agile Manifesto, referenced above, but they are inspired by it:

1. The full team owns their method, rather than a select few.
2. The focus is on method use rather than on comprehensive method description.
3. The team’s method evolves, rather than staying fixed.

These mindsets are crucial in your journey as a software professional. Not only must you understand technology, but you must also collaborate effectively with your teammates and lead your teams effectively. To help you do this well, we delve deeper into each of these mindset changes in the following sections.

---

1. Agile Manifesto, <http://agilemanifesto.org/principles.html>

## 23.2 The Full Team Owns Their Method

Traditionally, there is a dichotomy between method engineers and method users (i.e., developers):

1. Method engineers who define methods often do not use the methods themselves.
2. Method users who acquire real experiences with methods often are not asked, or do not have time, to give feedback and refine the methods they use.

As a result, method users often find method discussions to be out of touch with their real experiences and therefore a waste of time.

The kernel approach seeks to bridge this divide by placing a proper balance on all stakeholder perspectives. It recognizes the value of every team member in determining what ways of working are best for their team.

The Essence kernel can support this proper balancing through the use of the alphas, states, checklists, and by describing a team's own method as a composition of practices. Examples include:

1. conducting retrospectives guided by the alphas and their states;
2. defining practices on top of alphas and their states;
3. using alphas and states to agree on team member involvement and team responsibilities; and
4. using alphas and states to define lifecycles.

The alphas and their states and checklists provide a very simple but powerful tool, which team members can employ to take ownership of their own method.

## 23.3 Focus on Method Use

Traditionally, when most people talk about methods they are thinking in terms of method descriptions. Having a method description is a good thing in that it allows new team members or even existing team members to familiarize themselves with the team's method. Too often, however, these method descriptions fall short in communicating what team members really do in their day-to-day work. Unfortunately, the method descriptions have often become too heavyweight. This has only served to make the process description less useful, rather than more.

This dilemma is not best solved by more words, but rather by fewer words and more use. How do teams and team members actually use methods, then, to help

them in their day-to-day jobs? Consider the following needs with respect to a team's methods.

1. Teams need a way to determine real development progress.
2. Teams need to plan their endeavors and their sprints, and they need to discuss and agree upon what it means to be done.
3. Teams need to organize their team members, and agree on team member involvement and responsibilities.
4. Teams need to do their work and adapt their way of working.
5. Teams need to scale to varying size endeavors to handle varying challenges and complexity.

## 23.4 Evolve Your Team's Method

There is no one-size-fits-all when it comes to methods. This implies the following:

1. You cannot simply take any method and follow it blindly. All methods must be adapted to fit your situation.
2. Once adapted to your current situation, you are certain to learn more as your endeavor proceeds, requiring more adaptations.

A team's method is never fixed. Teams must constantly evolve their method as long as there is work to do on the product. This implies two fundamentals:

1. Always be ready to embrace new and better ways of working.
2. Always consider your current development situation when considering a change.

Evolving a method is straightforward with the kernel approach. You start with the kernel, and evaluate the practices you already have. Practices that are inadequate are then refined or replaced with better ones. This is best done gradually so continuous improvement becomes natural and not something you need to think about at great length.

Making continuous improvement a natural habit is easier said than done. It takes effort and proactive leadership and culture. So, on top of having a practice architecture, great organizations invest in creating a conducive environment to promote continuous improvement. Our approach of decomposing complex methods into much smaller practices helps teams take ownership of their methods and the outcomes of using them.

You have chosen a profession that is at the forefront of technological advances. Not many other professions are changing at a similar pace. To face such challenges, you need to be well prepared; what you have learned in this book, especially about Essence, will help you go a long way. As a final and parting note, we urge you to always be learning and growing, uncovering better ways of working all the time.

### What Should You Now Be Able to Accomplish?

After studying this chapter, you should be able to

- explain the intent of the Agile Manifesto and its key principles;
- describe mindset changes that need to happen when becoming agile;
- explain what a team really needs from a method, by giving examples; and
- explain what it means to say there is no one-size-fits-all when it comes to methods, and what this implies.

### Recommended Additional Reading

- R. Knaster and D. Leffingwell, *SAFe 4.0 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering* [Knaster and Leffingwell 2017]. This is a great resource for understanding the Scaled Agile Framework (SAFe). It covers SAFe 4.0.
- S. W. Ambler and M. Lines, *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise* [Ambler and Lines 2012]. This is an excellent resource for understanding Disciplined Agile Delivery (DAD).
- C. Larman and B. Vodde, *Large-Scale Scrum: More with LeSS* [Larman and Vodde 2016]. A great overview of LeSS, a method to apply Scrum in large-scale development.
- I. Jacobson, I. Spence, and P.-W. Ng. Is there a single method for the Internet of Things? [Jacobson et al. 2017].





## References

- Alpha State Card Games. 2018. <https://www.ivarjacobson.com/publications/brochure/alpha-state-card-games>. 99, 153
- S. Ambler and M. Lines. 2012. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press. 297, 339, 346
- K. Beck. 1999. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman. 203, 285, 346
- K. Beck. 2003. *Test-Driven Development by Example*. Addison Wesley. 346
- K. Bittner and I. Spence. 2003. *Use Case Modeling*. Addison-Wesley Professional, 2003. 222, 285
- G. Booch, J. Rumbaugh, and I. Jacobson. 2005. *The Unified Modeling Language User Guide*. 2nd edition. Addison-Wesley. 222, 285, 345
- F. Brooks. 1975. *The Mythical Man-Month*. Addison Wesley. 342
- M. Cohn. 2004. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional. 204, 247, 285
- E. Derby and D. Larsen. 2006. *Agile Retrospectives: Making Good Teams Great*. Pragmatic Bookshelf, Dallas, TX, and Raleigh, NC. 196, 198, 284
- E. W. Dijkstra. 1972. "The Humble Programmer." Turing Award Lecture, *CACM* 15 (10): 859–866. DOI: [10.1145/355604.361591](https://doi.org/10.1145/355604.361591). 342
- D. Graziotin and P. Abrahamsson. 2013. A web-based modeling tool for the SEMAT Essence theory of software engineering. *Journal of Open Research Software*, 1,1(e4); DOI: [10.5334/jors.ad.147](https://doi.org/10.5334/jors.ad.147), 153
- ISO/IEC/IEEE 2382. 2015. Information technology–Vocabulary. International Organization/International Electrotechnical Commission, Geneva, Switzerland. <https://www.iso.org/obp/ui/#iso:std:iso-iec:2382:ed-1:v1:en>. 343
- ISO/IEC/IEEE 12207. 2017. [https://en.wikipedia.org/wiki/ISO/IEC\\_12207](https://en.wikipedia.org/wiki/ISO/IEC_12207) 345
- ISO/IEC/IEEE 15288. 2002, 2008, 2015. Systems and software engineering—System life cycle processes. International Standardization Organization/International Electrotechnical Commission, 1 Rue de Varembe, CH-1211 Geneve 20, Switzerland. 345

- ISO/IEC/IEEE 24765. 2017. Systems and software engineering—Vocabulary. International Organization/International Electrotechnical Commission, Geneva, Switzerland. <https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:24765:ed-2:v1:en>. 343
- M. Jackson. 1975. *Principles of Program Design*. Academic Press. 344
- I. Jacobson. 1987. Object-oriented software development in an industrial environment. *Conference Proceedings of Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 87)*. DOI: [10.1145/38807.38824](https://doi.org/10.1145/38807.38824). 221, 285
- I. Jacobson and H. Lawson, editors. 2015. *Software Engineering in the Systems Context*, Systems Series, Volume 7. College Publications, London. 345
- I. Jacobson and E. Seidewitz. 2014. A new software engineering. *Communications of the ACM*, 12(10). DOI: [10.1145/2685690.2693160](https://doi.org/10.1145/2685690.2693160). 347
- I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach*. ACM Press Addison-Wesley. 345
- I. Jacobson, I. Spence, and K. Bittner. 2011. Use-Case 2.0: The Guide to Succeeding with Use Cases. <https://www.ivarjacobson.com/publications/whitepapers/use-case-ebook>. 169, 222, 226, 233, 285
- I. Jacobson, P.-W. Ng, P. E. McMahon, I. Spence, and S. Lidman. December 2012. The essence of software engineering: The SEMAT kernel. *Communications of the ACM*, 55(12). <http://queue.acm.org/detail.cfm?id=2389616>. DOI: [10.1145/2380656.2380670](https://doi.org/10.1145/2380656.2380670). 30, 95
- I. Jacobson, P.-W. Ng, P. E. McMahon, I. Spence, and S. Lidman. 2013a. *The Essence of Software Engineering: Applying the SEMAT Kernel*. Addison-Wesley. xxvi, 30, 90, 95, 336
- I. Jacobson, I. Spence, and P.-W. Ng. (October) 2013b. Agile and SEMAT: Perfect partners. *Communications of the ACM*, 11(9). <http://queue.acm.org/detail.cfm?id=2541674>. DOI: [10.1145/2524713.2524723](https://doi.org/10.1145/2524713.2524723). 30, 96
- I. Jacobson, I. Spence, and B. Kerr. 2016. Use-Case 2.0: The hub of software development. *Communications of the ACM*, 59(5): 61–69. DOI: [10.1145/2890778](https://doi.org/10.1145/2890778). 169, 222, 226, 285
- I. Jacobson, I. Spence, and P.-W. Ng. 2017. Is there a single method for the Internet of Things? *Queue*, 15.3: 20. DOI: [10.1145/3106637](https://doi.org/10.1145/3106637). 335, 339
- P. Johnson and M. Ekstedt. 2016. The Tarpit—A general theory of software engineering. *Information and Software Technology* 70: 181–203. [https://www.researchgate.net/profile/Pontus\\_Johnson/publication/278743539\\_The\\_Tarpit\\_-\\_A\\_General\\_Theory\\_of\\_Software\\_Engineering/links/55b4490008aed621de0114f5/The-Tarpit-A-General-Theory-of-Software-Engineering.pdf](https://www.researchgate.net/profile/Pontus_Johnson/publication/278743539_The_Tarpit_-_A_General_Theory_of_Software_Engineering/links/55b4490008aed621de0114f5/The-Tarpit-A-General-Theory-of-Software-Engineering.pdf). DOI: [10.1016/j.infsof.2015.06.001](https://doi.org/10.1016/j.infsof.2015.06.001). 91, 92, 96
- P. Johnson, M. Ekstedt, and I. Jacobson. September 2012. Where's the theory for software engineering? *IEEE Software*, 29(5). DOI: [10.1109/MS.2012.127](https://doi.org/10.1109/MS.2012.127). 84, 87, 96
- R. Knaster and D. Leffingwell. 2017. *SAFe 4.0 Distilled: Applying the Scaled Agile Framework for Lean Software and Systems Engineering*. Addison-Wesley Professional. 297, 339
- P. Kruchten. 2003. *The Rational Unified Process: An Introduction*. 3rd edition. Addison-Wesley. 345

- C. Larman and B. Vodde. 2008. *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum*. Pearson Education, Inc. 346
- C. Larman and B. Vodde. 2016. *Large-Scale Scrum: More with LeSS*. Addison-Wesley Professional. 297, 339
- D. Leffingwell. 2007. *Scaling Software Agility: Best Practices for Large Enterprises*. Addison-Wesley. 346
- P. E. McMahon. January/February 2015. A thinking framework to power software development team performance. *Crosstalk, The Journal of Defense Software Engineering*. <http://www.crosstalkonline.org/>. 96, 154
- NATO. 1968. "Software Engineering: Report on a conference sponsored by the NATO Science Committee." P. Naur and B. Randell, editors. Garmisch, Germany, October 7–11. 342
- S. Newman. 2015. *Building Microservices*. O'Reilly Media, Inc. 250, 285
- P.-W. Ng. 2013. Making software engineering education structured, relevant and engaging through gaming and simulation. *Journal of Communication and Computer* 10: 1365–1373. 99, 153
- P.-W. Ng. 2014. Theory based software engineering with the SEMAT kernel: Preliminary investigation and experiences. *Proceedings of the 3rd SEMAT Workshop on General Theories of Software Engineering*. ACM. DOI: 10.1145/2593752.2593756. 30, 96
- P.-W. Ng. 2015. Integrating software engineering theory and practice using Essence: A case study. *Science of Computer Programming*, 101: 66–78. DOI: 10.1016/j.scico.2014.11.009. 96, 152, 154
- Object Management Group. Essence—Kernel and Language for Software Engineering Methods (Essence). <http://www.omg.org/spec/Essence/1.1>. 63
- OMG Essence Specification. 2014. <http://www.omg.org/spec/Essence/Current>. 95, 346
- D. Ross. 1977. Structured Analysis (SA): A language for communicating ideas. In *IEEE Transactions on Software Engineering*, SE-3(1): 16–34. DOI: 10.1109/TSE.1977.229900. 344
- K. Schwaber and J. Sutherland. 2016. "The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game." Scrum.org.



# Index

- 201 Principles of Software Development*, 84–85
- Accept a User Story activity, 207, 214–215
- Acceptable state in Requirements alpha, 57, 215–217
- Acceptance Criteria Captured detail level for Test Case, 209
- Acceptance Criteria Listed detail level for Story Card, 209
- Achieving the Work alpha, 215
- ACM (Association for Computing Machinery), 342
- Actionability in Essence kernel, 89
- Activities
  - Essence, 55
  - Essence kernel, 72–75
  - Microservices Lite, 255–257, 267–270
  - Scrum, 175–176, 178
  - Scrum Lite, 188–197
  - thing to do, 62–63
  - Use Case Lite, 229, 238–244
  - User Story Lite, 207, 211–215, 217–218
- Activity spaces
  - Essence kernel, 68, 72–73
  - essentializing practices, 63–65
- Adapt in Plan-Do-Check-Adapt cycle, 132
- Adaptability in microservices, 252
- Adapts achievement level in Development competency, 61–62
- Addressed state in Requirements alpha, 57, 80
- Agile Manifesto, 335–336
- Agility and Agile methods
  - Agile methods era, 25–26
  - Essence kernel relationship to, 90–91
  - introduction, 346
  - practices and methods, 335–336
- All Stories Fulfilled state in Use Case alpha, 231
- Alphas
  - Chasing the State game, 105–108
  - Checkpoint Construction game, 112–113
  - composition of practices, 279–281
  - customer area of concern, 160–163
  - development, 128–132
  - development journey, 146–148
  - Essence, 54–55
  - Essence kernel, 68–72, 89, 151–152
  - kick-starting development, 122–123
  - large and complex development, 311
  - Microservices Lite, 257–259
  - Objective Go game, 108–111
  - overview, 56
  - Progress Poker game, 100–104
  - Scrum, 175–177
  - Scrum Lite, 179–182
  - states, 55–59
  - sub-alphas, 124
  - Use Case Lite, 229–233
  - User Story Lite, 207–209, 215–216
- Alternative practices, 278–279
- Ambler, Scott, 346
- Analysis competency, 76

- Analyzed state in Use-Case Slice alpha, 233
- Anomalies in development journey, 148
- Application logic, definition, 251
- Applies achievement level in Development competency, 61–62
- Architecture Selected state in Software Systems, 59, 80, 311
- Areas of concern in Essence kernel, 67–68
- Assists achievement level in Development competency, 61–62
- Association for Computing Machinery (ACM), 342
- Attainable attribute in SMART criteria, 196–197
- Automated detail level
  - Build and Deployment Script, 265
  - Test Case, 210
- AXE telecommunication switching system, 344
  
- Babbage, Charles, 341
- Background in practices, 34
- Backlog-Driven Development practice, 33
- Beck, Kent, 86, 346
- Booch, Grady, 345
- Bounded state in Requirements alpha, 56, 80, 215–216
- Briefly Described detail level in Use-Case Narrative work product, 235
- Brooks, Frederick P., 84, 342
- Build and Deployment Script, 264–265
- Building blocks, 10
- Bulleted Outline detail level in Use-Case Narrative work product, 235
- Bureau of the Census, 341
  
- Capabilities in practices, 33–34
- Capability Maturity Model Integration (CMMI), 306
- Capacity Described work product, 183
- Card games
  - Chasing the State, 105–108
  - Checkpoint Construction, 111–113
  - Objective Go, 108–111
  - overview, 97–99
  - Progress Poker, 99–105
  - reflection, 113
- Cards
  - alphas, 57–58
  - Essence, 38
  - user stories, 204
- Census, 341
- Chasing the State game, 105–108
- Check in Plan-Do-Check-Adapt cycle, 131–132
- Checking in Essence, 138–139
- Checkpoint Construction game, 111–113
- Checkpoint pattern, 78–80
- Checkpoints
  - kick-starting development, 122–123
  - kick-starting development with practices, 159–165
  - large and complex development, 310
- Cloud computing for microservices, 252
- CMMI (Capability Maturity Model Integration), 306
- COBOL programming language, 342
- Code
  - thing to work with, 54, 56
  - work product cards, 60
- Coder role pattern card, 79
- Coherent state in Requirements alpha, 57, 215–216
- Collaboration
  - importance, 11–12
  - Scrum, 165–166, 174
- Collaborations and Interfaces Defined detail level in Design Model work product, 261
- Common ground in Essence, 34–37
- Competencies
  - Essence, 55
  - Essence kernel, 68, 75–77
  - programming, 61–62
  - testing in, 10
- Compilers, 341–342
- Complete state in Microservices alpha, 258–259

- Completed PBIs Listed work product, 184
- Complex development. *See* Large and complex development
- Component methods, 22–25
- Component paradigm, 344–345
- Composition of practices
  - description, 276–282
  - Essence, 282–284
  - overview, 275–276
  - reflection, 282–283
- Conceived state in Requirements alpha, 56, 311
- Confirmation in user stories, 205
- Consensus-based games
  - Chasing the State, 105–108
  - Checkpoint Construction, 111–113
  - Objective Go, 108–111
  - Progress Poker, 99–105
  - reflection, 113
- Containers definition, 251
- Context in kick-starting development, 118–121
- Continual improvement in large and complex development, 321–323
- Continuous detail level in Build and Deployment Script, 265
- Conversation Captured detail level in Story Card, 209
- Conversations in user stories, 205
- Coordination in activity space, 74
- Culture issues, 330–331
- Customer area of concern
  - alphas, 160–163
  - competencies, 76
  - development perspective, 119–120
  - development process, 139
  - Essence kernel, 68–70
- Customer-related practices, 19
- Customers
  - description, 42–43
  - value for, 43–44
- DAD (Disciplined Agile Delivery)
  - agile scaling, 27
  - introduction, 346
  - monolithic methods, 297
  - practices from, 296
- Daily Scrum activity
  - description, 173, 178
  - diagram, 175–176
  - overview, 192–193
- Daily Standup practice in Scrum, 26, 33, 173
- Data in structured methods era, 21–22
- Data processing focus, 341
- Data stores, definition, 251
- Davis, Alan, 84–85
- Definition of Done (DoD) in Scrum, 176–177
- Demonstrable alpha state, 59, 100
- Deployment in activity space, 74
- Descriptive theory of software engineering, 87–88
- Design Model work product, 254, 257, 260–263
- Design overview, 14
- Design Patterns Identified detail level in Design Model work product, 262
- Design phase
  - iterative method, 21
  - waterfall method, 19–20
- Detail levels
  - Build and Deployment Script, 265
  - Microservice Design work product, 264
  - Story Card, 209
  - Test Case, 210
  - Use Case Lite work products, 233–236
  - Use-Case Narrative work product, 235
  - Use-Case Slice Test Case work product, 237
  - work products, 60–61
- Developers
  - Scrum, 173
  - Tarpit theory, 92
- Development
  - doing and checking, 138–139
  - kick-starting. *See* Kick-starting development; Kick-starting development with practices
  - overview, 127–132

- Development (*continued*)
  - Plan-Do-Check-Adapt cycle, 128–132
    - plans, 132–138
    - way of working, 140–142
- Development competency, 61–62, 77
- Development Complete checkpoint, 80
- Development endeavor, 79–80
- Development journey
  - anomalies, 148
  - overview, 145
  - progress and health, 146–148
  - visualizing, 145–146
- Development types
  - culture issues, 330–331
  - overview, 325
  - practice and method architectures, 326–328
  - practice libraries, 328–330
- DevOps practice, 302
- Dijkstra, E. W., 86, 342–343
- Disciplined Agile Delivery (DAD)
  - agile scaling, 27
  - introduction, 346
  - monolithic methods, 297
  - practices from, 296
- Disciplined approach in software engineering, 14–15
- Do in Plan-Do-Check-Adapt cycle, 131
- Document elements in Essence, 54
- DoD (Definition of Done) in Scrum, 176–177
- Doing alpha in PBIs, 181
- Doing in development, 138–139
- Done alpha in PBIs, 181
- Done term, definition, 99–100
- EA (enterprise architecture), 24
- Endeavor area of concern
  - competencies, 77
  - development perspective, 120–121
  - development process, 136–139
  - Essence kernel, 68, 70–71
  - kick-starting development with practices, 163–165
  - practices, 19
  - Scrum, 199
- Endeavors
  - description, 42–43
  - teams, 48–49
  - ways of working in, 49–50
  - work in, 49
- Engaging user experiences, 37–38
- Enterprise architecture (EA), 24
- Ericsson AB, 344
- Essence
  - common ground, 34–37
  - composition of practices, 282–284
  - development. *See* Development
  - development journey. *See* Development journey
  - engaging user experiences, 37–38
  - essentializing practices, 63–65
  - essentials focus, 37
  - evolution, 346
  - insights, 32
  - kick-starting development. *See* Kick-starting development
  - language, 54–61
  - large and complex development, 310–311, 322–324
  - methods and practices, 32–34
  - microservices, 252–256
  - OMG standard, 29–30
  - overview, 31
  - practices, 298–299
  - purpose, 42
  - Scrum with, 174–179
  - serious games. *See* Serious games
  - theory of software engineering, 87–91
  - Use Case Lite practice, 227–230
  - User Story Lite practice, 207–208
  - work products, 60
- Essence kernel
  - actionability, 89
  - activities, 72–75
  - alphas, 68–72
  - applying, 151–152
  - competencies, 75–77
  - extensibility, 90



- growth from, 93–94
  - observations, 151
  - organizing with, 67–69
  - overview, 67
  - patterns, 77–80
  - practicality, 88–89
  - relationship to other approaches, 90–91
  - User Story Lite practice, 215–218
  - validity, 151
- Essential Outline detail level in Use-Case
  - Narrative work product, 235
- Essentialized practices, 35–36
- Essentializing practices
  - composition of practices, 283–284
  - description, 35–36
  - Essence, 298–299
  - for libraries, 329
  - monolithic methods and fragmented practices, 296–298
  - overview, 63–65
  - reusable, 299–302
  - sources, 295–296
- Estimatable criteria in user stories, 205–206
- Evolve Microservice activity, 255, 257, 269–270
- Exchangeable packages, 345
- Explicit approaches in Scrum, 173–174
- Extensibility
  - Essence kernel, 90
  - software systems, 47
- Extension practices, 279
- Extreme Programming Explained*, 86
- Extreme Programming (XP)
  - introduction, 346
  - practices from, 296
  - user stories, 203
- Feedback in Use Case Lite practice, 239
- Find Actors and Use Cases activity, 229, 238–239
- Find User Stories activity, 207, 212
- Formed state in Teams, 311
- Fortran programming language, 342
- Foundation Established state in Way of working, 311
- Fragmented practices, 296–298
- Fulfilled alpha state, 57
- Fully Described detail level in Use-Case
  - Narrative work product, 235
- Function-data paradigm, 344
- Functionality in software systems, 46
- Functions in structured methods era, 21–22
- Future, dealing with
  - agility, 335–336
  - methods evolution, 338–339
  - methods use, 337–338
  - overview, 333–335
  - teams and methods, 337
- Games
  - Chasing the State, 105–108
  - Checkpoint Construction, 111–113
  - Objective Go, 108–111
  - overview, 97–99
  - Progress Poker, 99–105
  - reflection, 113
- General predictive theory of software engineering, 91–92
- “Go to statement considered harmful”
  - article, 86
- Goal Established state in Use Case alpha, 230
- Goals Specified work product, 183
- Gregor, Shirley, 84–85
- Hacking vs. programming, 6
- Handle favorites use-case slice, 242–243
- Happy day scenarios, 224
- Health and progress
  - development journey, 146–148
  - Essence, 54
  - Microservices Lite, 271–272
  - use-case slices, 245–246
- Hemdal, Göran, 344
- Higher-level languages, 342
- History of software and software engineering, 341–347

- Hollerith punched card equipment, 341
- Hopper, Grace Murray, 341–342
- Identification of microservices, 251–252
- Identified state
  - Microservices Lite, 258
  - user stories, 209
- Identify Microservices activity, 255, 257, 267–268
- IEEE (Institute of Electrical and Electronic Engineering), 342
- Implementation phase
  - activity space, 74
  - iterative method, 21
  - waterfall method, 19–20
- Implemented state in Use-Case Slice alpha, 233
- In Progress state in user stories, 209
- Increment elements
  - description, 177
  - work products, 183–184
- Increment Notes Described work product, 184
- Incremental development in use cases
  - slices, 226–227
- Independent criteria in user stories, 205
- Innovates achievement level in Development competency, 61–62
- Institute of Electrical and Electronic Engineering (IEEE), 342
- Interfaces Specified detail level in
  - Microservice Design work product, 264
- Internal Elements Designed detail level in
  - Microservice Design work product, 264
- Internal Structure Defined detail level in
  - Microservice Design work product, 264
- INVEST criteria for user stories, 205–206
- ISO/IEC 12207 standard, 345
- Items Ordered work product, 182
- Iterative operations
  - development, 127
  - development journey, 147
  - large and complex development, 319–321
  - lifecycle methods, 20–21
- Jackson, Michael, 344
- Jackson Structured Programming (JSP), 344
- Jacobson, Ivar
  - component paradigm, 344
  - method prison governing, 27
  - OMG, 345
  - RUP, 345
  - SEMAT, 28, 346
  - Use-Case Driven Development practice, 221
- JSP (Jackson Structured Programming), 344
- Kernel. *See* Essence kernel
- Key elements of software engineering
  - basics, 41–43
  - endeavors, 48–50
  - overview, 41
  - value for customers, 43–45
  - value through solutions, 45–48
- Kick-starting development
  - context, 118–121
  - overview, 117–118
  - scope and checkpoints, 122–123
  - things to watch, 124–126
- Kick-starting development with practices
  - context, 158–159
  - overview, 157–158
  - practices to apply, 165–167
  - scope and checkpoints, 159–165
  - things to watch, 167–169
- Kruchten, Philippe
  - method prison governing, 27
  - RUP, 345
- Language of software engineering
  - competencies, 61–62
  - essentializing practices, 63–65
  - overview, 53
  - practice example, 53–54
  - things to do, 62–63

- things to work with, 54–61
- Large and complex development
  - alphas, 311
  - common vision, 315–317
  - continual improvement, 321–323
  - Essence, 310–311, 322–324
  - iterative operations, 319–321
  - kick-starting, 309–315
  - large-scale development, 308–309
  - large-scale methods, 306–308
  - managing, 317–319
  - overview, 305–306
  - practices, 310–313
  - running, 315–322
  - scope and checkpoints, 310
  - things to watch, 313–315
- Large-scale integrated circuits, 343
- Large-Scale Scrum (LeSS)
  - agile scaling, 27
  - introduction, 346
  - monolithic methods, 297
  - practices from, 296
- Larman, Craig, 346
- Lawson, Harold “Bud,” 345
- Leadership competency, 77
- Leffingwell, Dean, 346
- LeSS (Large-Scale Scrum)
  - agile scaling, 27
  - introduction, 346
  - monolithic methods, 297
  - practices from, 296
- Levels of detail
  - Build and Deployment Script, 265
  - Microservice Design work product, 264
  - Story Card, 209
  - Test Case, 210
  - Use Case Lite work products, 233–236
  - Use-Case Narrative work product, 235
  - Use-Case Slice Test Case work product, 237
  - work products, 60–61
- Libraries for practices, 328–330
- Lifecycles, 19–21
- Lines, Mark, 346
- Lovelace, Ada, 341
- Machine instruction level, 341
- Make Evolvable activity, 255, 257, 268–269
- Management competency, 77
- Martin, Robert, 90
- Masters achievement level in Development
  - competency, 61–62
- Mayer, Bertrand, 28
- Measurable attribute in SMART criteria, 196–197
- Method prison, 27
- Methods
  - agile methods era, 25
  - component methods era, 22–25
  - consequences, 26–28
  - definition, 19
  - Essence, 32–34
  - evolution, 338–339
  - large-scale, 306–308
  - lifecycles, 19–21
  - people practices, 25–26
  - rise of, 18–19
  - structured methods era, 21–22
  - team ownership, 337
  - technical practices, 21–25
  - use focus, 337–338
- Methods war, 22, 26–27
- Meyer, Bertrand, 346
- Microprocessors, 343
- Microservice alpha, 254, 257
- Microservice Build and Deployment work
  - product, 254, 257
- Microservice Design work product, 254, 257, 263–264
- Microservice Test Case work product, 255, 257, 265–267
- Microservices, 166–169
  - description, 250–252
  - Essence, 252–256
  - overview, 249–250
- Microservices Lite practice
  - activities, 255–256, 267–270
  - alphas, 257–259

- Microservices Lite practice (*continued*)
  - Build and Deployment Script, 264–265
  - description, 256–257
  - design model, 260–263
  - impact, 270–271
  - Microservice Design work product, 263–264
  - Microservice Test Case work product, 265–267
  - overview, 253–256
  - progress and health, 271–272
  - reusable practices, 299–300
  - work products, 259–267
- Mini-computers, 343
- Mini-methods, 19
- Minimal state in Microservices Lite, 258
- Modular approaches in Scrum, 173–174
- Monolithic methods, 296–298
- Mythical Man-Month*, 84
- NATO-sponsored conference, 342
- Negotiable criteria in user stories, 205
- NZ Transport Agency, 18
- Object Management Group (OMG) standard
  - Essence, 29–30, 346
  - Essence kernel, 71
  - notation, 23–24
  - UML standard, 345
- Object-oriented programming
  - acceptance, 344–345
  - components in, 23
- Objective Go game, 108–111
- On the Criteria to Be Used in Decomposing Systems into Modules*, 86
- Operational alpha state, 59
- Opportunity
  - alpha state card, 72
  - customer area of concern, 69, 71
  - development context, 158
  - development endeavors, 42–43
  - development perspective, 119–120
  - development plans, 133–134
  - large and complex development, 311–312
  - scope and checkpoints, 161–162
  - value for customers, 43–44
- Outlined detail level in Build and Deployment Script, 265
- Pair programming teams, 26
- Paradigm shifts, 22–23
- Paradigmatic theories, 85
- Paths in use cases slices, 228
- Patterns
  - Essence kernel, 68, 77–80
  - essentializing practices, 63–65
  - Scrum, 178–179, 184–186
- PBIs. *See* Product Backlog Items (PBIs)
- People practices, 25–26
- Performance in software systems, 47
- Perlis, Alan, 92
- PLA (product-line architecture), 24
- Plan-Do-Check-Adapt cycle, 128–132
- Planned alpha in sprints, 179
- Plans
  - development, 132–138
  - Plan-Do-Check-Adapt cycle, 128–131
  - Scrum Lite, 188–192
- POs (product owners)
  - description, 178
  - pattern cards, 184–185
  - Scrum, 172–173, 175
- Possibilities in activity space, 73
- Post-development phase in development endeavor, 79–80
- Practicality in Essence kernel, 88–89
- Practice separation in Essence kernel, 90
- Practices
  - agile methods era, 25
  - background, 34
  - capabilities, 33–34
  - common ground, 34–37
  - component methods era, 22–25
  - composition of. *See* Composition of practices
  - consequences, 26–28
  - definition, 174
  - Essence, 32–34, 298–299

- fragmented practices, 296–298
- kick-starting development with. *See* Kick-starting development with practices
- large and complex development, 310–313
- libraries, 328–330
- lifecycles, 19–21
- people, 25–26
- reusable, 299–302
- rise of, 18–19
- Scrum, 173–174, 177, 198–199
- sources, 295–296
- structured methods era, 21–22
- technical, 21–25
- types, 19
- Pre-development phase in development endeavor, 79–80
- Precision in Scrum, 200–202
- Preparation in activity space, 74
- Prepare a Use-Case Slice activity, 229, 242–243
- Prepare a user story activity, 207, 212–213
- Prepared state
  - Use-Case Slice alpha, 232–233
  - Work alpha, 215
- Priorities in Scrum, 172
- Problems in kick-starting development, 118
- Product Backlog Items (PBIs)
  - alphas, 181
  - description, 172, 177
  - example, 176
  - identifying, 173
  - Scrum, 168
- Product Backlog practice, 302
- Product Backlog work product
  - activity cards, 190–192
  - description, 177
  - Scrum Lite, 182–184
- Product-line architecture (PLA), 24
- Product Management practice, 302
- Product owners (POs)
  - description, 178
  - pattern cards, 184–185
  - Scrum, 172–173, 175
- Product Ownership practice, 301
- Product Retrospective practice, 302
- Product Sprint practice, 302
- Program backlog management, 318
- Program practices, 302–303
- Programming, defined, 4
- Programming and software engineering
  - differences, 6–8
  - intern view, 8–10
  - overview, 3–4
  - professional view, 10–12
  - programming, 4–6
  - software engineering, 12–15
- Progress and health
  - activity space, 74–75
  - development journey, 146–148
  - Essence, 54
  - Microservices Lite practice, 271–272
  - use-case slices, 245–246
- Progress Poker game
  - benefits, 102
  - example, 103–105
  - overview, 99–102
- Progressing
  - use-case slices, 232–233
  - use cases, 230–232
- Provided interface, UML notation for, 261
- Quality in software systems, 47–48
- Quantifiable approach in software engineering, 14–15
- Rapidly Deployable state in Microservices Lite practice, 258
- Rational Unified Process (RUP)
  - development of, 24, 345
  - large-scale development, 306
  - monolithic methods, 297
- Reaching out in scaling, 293
- Ready for Development checkpoint, 80
- Ready for Development state in User Story, 209
- Ready requirement, 80

- Ready state
  - PBIs, 181
  - Software Systems, 59
- Recognized state for Stakeholders, 311
- Relevant attribute in SMART criteria, 196–197
- Reliability in software systems, 47
- Required Behavior Defined detail level in Microservice Design work product, 264
- Required interface, UML notation for, 261
- Requirements
  - activity space, 74
  - alpha state card, 72
  - alphas, 56–58
  - development context, 158
  - development perspective, 120
  - development plans, 134–135
  - large and complex development, 311–312
  - Ready for Development checkpoint, 80
  - scope and checkpoints, 161–162
  - solution area of concern, 70
  - in solutions, 42–43, 45–46
  - thing to work with, 54–56
  - User Story Lite practice, 225, 227, 230
- Requirements alpha
  - Progress Poker game, 100–101
  - User Story Lite practice, 215–217
- Requirements engineering, 13–14
- Requirements phase
  - iterative method, 21
  - waterfall method, 19–20
- Retired alpha state, 59
- Retrospective practice in Scrum, 33
- Reusable practices, 19, 299–302
- Reviewed alpha in sprints, 179–180
- Roles in Scrum Lite, 184–186
- Roles pattern, 77–78
- Ross, Douglas, 344
- Royce, Walker, 345
- Rumbaugh, James, 345
- RUP (Rational Unified Process)
  - development of, 24, 345
  - large-scale development, 306
  - monolithic methods, 297
- SA/SD (Structured Analysis/Structured Design), 21
- SaaS (Software as a Service), 8
- SADT (Structured Analysis and Design Technique)
  - description, 21–22
  - development of, 344
- Scaled Agile Framework (SAFe)
  - agile scaling, 27
  - introduction, 346
  - monolithic methods, 297
  - practices from, 296
- Scaled Professional Scrum (SPS)
  - agile scaling, 27
  - introduction, 346
  - practices from, 296
- Scaling
  - challenges, 289–291
  - dimensions of, 291–294
  - large and complex development. *See* Large and complex development
  - overview, 289
  - reaching out, 293
  - scaling up, 292–293
  - zooming in, 291–292
- Scenario Chosen detail level in Use-Case Slice Test Case work product, 237
- Scenarios in use cases slices, 228
- Scheduled alpha in sprints, 179
- Schwaber, Ken, 346
- Scope
  - kick-starting development, 122–123
  - kick-starting development with practices, 159–165
  - large and complex development, 310
- Scoped state in Use-Case Slice alpha, 232
- Scripted detail level in Test Case, 210
- Scripted or Automated detail level in Use-Case Slice Test Case work product, 237
- Scrum
  - collaboration, 165–166, 174

- components, 33
- composite practices, 306–307
- description, 168
- with Essence, 174–179
- fragmented practices, 297
- introduction, 346
- overview, 171–173
- practices, 173–174, 198–199, 296
- precision, 200–202
- reflections, 198–202
- Scrum Lite
  - activities, 188–197
  - alphas, 179–182
  - overview, 174–177
  - planning, 188–192
  - roles, 184–186
  - usage, 187–188
  - work products, 182–184
- Scrum Masters
  - description, 173, 178–179
  - large and complex development, 321–322
  - pattern cards, 184–186
  - patterns, 175
- Scrum of Scrums meetings, 320
- Scrum Teams
  - description, 179
  - Essence, 175
  - pattern cards, 185–186
- SDL (Specification and Description Language), 344
- Self-organizing teams, 26
- SEMAT (Software Engineering Method And Theory)
  - description, 28–29
  - founding, 346
- Serious games
  - Chasing the State, 105–108
  - Checkpoint Construction, 111–113
  - Objective Go, 108–111
  - overview, 97–99
  - Progress Poker, 99–105
  - reflection, 113
- Service-oriented architecture (SOA), 24
- Simplest Story Fulfilled state in Use Case
  - alpha, 231
- Simula 67 language, 23
- Slice the Use Cases activity
  - description, 229
  - working with, 241–242
- Slicing use cases, 226–227
- Small attribute
  - SMART criteria, 196–197
  - user stories, 206
- Smalltalk language, 23
- SMART criteria, 196–197
- “So that” clauses in user stories, 206
- SOA (service-oriented architecture), 24
- Social issues, 330–331
- Software as a Service (SaaS), 8
- Software crisis, 18, 343
- Software development, defined, 4
- Software Engineering Method And Theory (SEMAT)
  - description, 28–29
  - founding, 346
- Software engineering overview
  - challenges, 17–18
  - defined, 4–5, 14–15
  - history, 341–347
  - key elements. *See* Key elements of software engineering
  - language. *See* Language of software engineering
  - methods and practices, 18–28
  - OMG standard, 29–30
  - and programming. *See* Programming and software engineering
  - SEMAT initiative, 28–29
  - Tarpit theory, 92
  - theory, 84–87
- Software Life Cycle Processes, 345
- Software Systems
  - alpha cards, 58, 72
  - Demonstrable alpha state card, 100
  - development context, 158
  - development perspective, 120
  - development plans, 135–136

- Software Systems (*continued*)
  - large and complex development, 311–312
  - Objective Go game, 109–111
  - scope and checkpoints, 161, 162
  - solutions, 42–43, 45–48, 70
  - thing to work with, 54–56
- Soley, Richard, 28, 346
- Solution area of concern
  - competencies, 76–77
  - development perspective, 120
  - development process, 139
  - Essence kernel, 68–70
  - kick-starting development with practices, 161
- Solution-related practices, 19
- Solutions
  - description, 42–43
  - value through, 45–48
- Specification and Description Language (SDL), 344
- Splitting User Stories activity, 207, 213–214
- Sprint Backlog
  - activity cards, 190–191
  - description, 177
  - PBIs, 172
  - work products, 183
- Sprint Planning activity
  - activity cards, 188–192
  - description, 178
- Sprint Retrospective activity
  - activity cards, 195–196
  - Scrum, 178
- Sprint Review activity
  - activity cards, 193–195
  - description, 172, 178
- Sprints
  - alphas, 179–181
  - description, 177
  - Scrum, 172–173
- SPS (Scaled Professional Scrum)
  - agile scaling, 27
  - introduction, 346
  - practices from, 296
- Stakeholder alpha in Chasing the State game, 105–107
- Stakeholder Representation competency, 76
- Stakeholders
  - activity space, 74
  - alpha state card, 72
  - customer area of concern, 69, 71
  - as customers, 42–43
  - development context, 158
  - development perspective, 119
  - development plans, 133
  - large and complex development, 311–312
  - Objective Go game, 108–111
  - scope and checkpoints, 159–160
  - value for, 44–45
- Started state in Work alpha, 311
- States in alphas, 55–59
- Stored program computers, 341
- Story Card work product, 207, 209–210
- Story practice, 166
- Story Structure Understood state in Use Case alpha, 231
- Structure and Approach Described detail level in Design Model work product, 260
- Structured Analysis and Design Technique (SADT)
  - description, 21–22
  - development of, 344
- Structured Analysis/Structured Design (SA/SD), 21
- Structured detail level in Use-Case Model work product, 235
- Structured methods era, 21–22
- Student Pairs pattern card, 78
- Sub-alphas, 124
- Subsystems in UML notation, 261
- Sufficient Stories Fulfilled state in Use Case alpha, 231
- Support in activity space, 74–75
- Sutherland, Jeff, 346
- SWEBOK, 84–85
- System Boundary Established detail level in Use-Case Model work product, 234
- Systematic approach in software engineering, 14–15



- Tarpit theory, 91–92
- TD (test-driven development) in Essence, 36
- TDD (Test-Driven Development) in Extreme Programming, 346
- Team Backlog practice, 301
- Team Retrospective practice
  - description, 301
  - large and complex development, 321–322
- Team Sprint practice, 301
- Teams
  - activity space, 74–75
  - agile, 26
  - alpha state card, 72
  - development perspective, 120
  - development plans, 136–137
  - endeavor area of concern, 42–43, 48–49, 70–71
  - Essence, 36
  - large and complex development, 311–312
  - methods ownership, 337
  - need for, 12–13
  - scope and checkpoints, 163–164
- Technical practices, 21–25
- Technology stacks, 10
- Test a Use-Case Slice activity
  - description, 229
  - working with, 243–244
- Test Automated detail level in Microservice
  - Test Case work product, 267
- Test Case work product, 207, 209–210
- Test Dependencies Managed detail level in Microservice Test Case work product, 266
- Test-driven development (TD) in Essence, 36
- Test-Driven Development (TDD) in Extreme Programming, 346
- Test Scenarios Chosen detail level in Microservice Test Case work product, 266
- Testable attribute
  - SMART criteria, 196–197
  - user stories, 206
- Testing
  - activity space, 74
  - waterfall method phase, 19–20
- Testing competency, 10, 77
- Theory
  - arguments, 85–87
  - Essence, 87–91
  - general predictive theory, 91–92
  - growth from, 93–94
  - overview, 83–84
  - software engineering, 84–87
  - uses, 87
- Things to do
  - activities, 62–63
  - backlogs, 49
  - composition, 279
  - Essence kernel, 72–75
- Things to watch
  - kick-starting development, 124–126
  - kick-starting development with practices, 167–169
  - large and complex development, 313–315
- Things to work with
  - alpha states, 56–59
  - alphas, 56
  - Essence kernel, 69–72, 89
  - overview, 54–56
  - Use Case Lite practice, 230–234
  - work products, 59–61
- To Do alpha in PBIs, 181
- Turing tar-pit, 92
- UCDD (Use-Case Driven Development)
  - practice, 221–222
- Unified Modeling Language (UML) standard
  - development of, 24
  - introduction, 345
  - Microservices Lite practice, 260–261
  - primer, 260
  - use cases, 222–223
- Unified Process prison, 27
- Unified Process (UP), 24, 345
- Univac I computer, 341
- University of Wisconsin, 18
- UP (Unified Process), 24, 345
- Usable alpha state, 59
- Use Case alpha, 229–231

- Use-Case diagrams, 24
- Use-Case Driven Development (UCDD)
  - practice, 221–222
- Use Case Lite practice
  - activities, 238–244
  - alphas, 229–233
  - Essence, 227–230
  - impact, 244–245
  - kick-starting, 237–240
  - overview, 221–222
  - reusable practices, 299–300
  - use-case slices progress and health, 245–246
  - use cases description, 222–226
  - use cases slicing, 226–227
  - user stories vs. use cases, 246–248
  - work products, 233–236
  - working with, 240–244
- Use-Case Model work product, 227, 229, 234–235
- Use-Case Narrative work product, 227, 229, 235–236
- Use-case narratives, 224–225
- Use case practices, 166, 168–169
- Use-Case Slice alpha, 229, 232–233
- Use-Case Slice Test Case work product, 227, 229, 236–237
- Use-case slices
  - process, 226–227
  - progress and health, 245–246
- Use Cases
  - introduction, 345
  - practices from, 296
- User experiences in Essence, 37–38
- User interface, definition, 251
- User stories
  - description, 204–207
  - Scrum teams, 166
- User Stories practice
  - description, 168–169
  - vs. use cases, 246–248
- User Story alpha in User Story Lite practice, 207–208
- User Story for Extreme Programming, 346
- User Story Lite practice
  - activities, 211–215
  - alphas, 207–209
  - Essence, 207–208
  - Essence kernel, 215–218
  - impact, 216–218
  - overview, 203
  - usage, 211
  - user story description, 204–207
  - work products, 209–210
- Validity in Essence kernel, 151
- Valuable criteria in user stories, 205
- Value
  - for customers, 43–45
  - through solutions, 45–48
- Value Established detail level in Use-Case Model work product, 234
- Value Established state in Opportunity, 311
- Value Expressed detail level in Story Card, 209
- Variables Identified detail level in Use-Case Slice Test Case work product, 237
- Variables Set detail level in Use-Case Slice Test Case work product, 237
- Verification phase
  - iterative method, 21
  - waterfall method, 19–20
- Verified state
  - Use-Case Slice alpha, 233
  - user stories, 209
- Vodde, Bas, 346
- von Neumann, John, 341
- Waterfall method
  - description, 19–20
  - development of, 344
- Way of working
  - adapting, 140–141
  - alpha state card, 72
  - development context, 158
  - development perspective, 120–121
  - development plans, 138

- endeavor area of concern, 42–43, 49–50, 71
- Essence kernel, 141–142
- large and complex development, 311–312
- scope and checkpoints, 163, 165
- “Where’s the Theory for Software Engineering?” paper, 84
- Work activity
  - alpha state card, 72
  - development context, 158
  - development perspective, 120
  - development plans, 136–137
  - endeavor area of concern, 42–43, 49, 71
  - large and complex development, 311–312
  - scope and checkpoints, 163–164
- Work alpha, 215–216
- Work Forecast Described work product, 183
- Work products
  - Essence, 54–55
  - Microservices Lite practice, 259–267
  - overview, 59–61
  - Scrum, 175, 177
  - Scrum Lite, 182–184
  - Use Case Lite practice, 229, 233–236
  - User Story Lite practice, 207, 209–210
- Write Code activity cards, 62–63
- XP (Extreme Programming)
  - introduction, 346
  - practices from, 296
  - user stories, 203
- Zooming in in scaling, 291–292



## Author Biographies

### Ivar Jacobson



**Dr. Ivar Jacobson** received his Ph.D. in computer science from KTH Royal Institute of Technology, was awarded the Gustaf Dalén medal from Chalmers in 2003, and was made an honorary doctor at San Martin de Porres University, Peru, in 2009. Ivar has both an academic and an industry career. He has authored ten books, published more than a hundred papers, and is a frequent keynote speaker at conferences around the world.

Ivar Jacobson is a key founder of components and component architecture, work that was adopted by Ericsson and resulted in the greatest commercial success story ever in the history of Sweden (and it still is). He is the creator of use cases and Objectory—which, after the acquisition of Rational Software around 2000, resulted in the Rational Unified Process, a popular method. He is also one of the three original developers of the Unified Modeling Language. But all this is history. His most recently founded company, Ivar Jacobson International, has been focused since 2004 on using methods and tools in a smart, superlight, and agile way. Ivar is also a founder and leader of a worldwide network, SEMAT, whose mission is to revolutionize software development based on a kernel of software engineering. This kernel has been realized as a formal standard called Essence, which is the key idea described in this book.

## Harold “Bud” Lawson



**Professor Emeritus Dr. Harold “Bud” Lawson** (The Institute of Technology at Linköping University) has been active in the computing and systems arena since 1958 and has broad international experience in private and public organizations as well as academic environments. Bud contributed to several pioneering efforts in hardware and software technologies. He has held professorial appointments at several universities in the USA, Europe, and the Far East. A Fellow of the ACM, IEEE, and INCOSE, he was also head of the Swedish delegation to ISO/IEC JTC1 SC7 WG7 from 1996 to 2004 and the elected architect of the ISO/IEC 15288 standard. In 2000, he received the prestigious IEEE Computer Pioneer Charles Babbage medal award for his 1964 invention of the pointer variable concept for programming languages. He has also been a leader in systems engineering. In 2016, he was recognized as a Systems Engineering Pioneer by INCOSE. He has published several books and was the coordinating editor of the “Systems Series” published by College Publications, UK.

Tragically, Harold Lawson passed away after battling an illness for almost a year, just weeks before the publication of this book.

## Pan-Wei Ng



**Dr. Pan-Wei Ng** has been helping software teams and organizations such as Samsung, Sony, and Huawei since 2000, coaching them in the areas of software development, architecture, agile, lean, DevOps, innovation, digital, Beyond Budgetings, and Agile People. Pan-Wei firmly believes that there is no one-size-fits-all, and helps organizations find a way of working that suits them best. This is why he is so excited about Essence and has been working with it through SEMAT since their inception in 2006, back when Essence was a mere

idea. He has contributed several key concepts to the development of Essence.

Pan-Wei coauthored two books with Dr. Ivar Jacobson and frequently shares his views in conferences. He currently works for DBS Singapore, and is also an adjunct lecturer in the National University of Singapore.

## Paul E. McMahon



**Paul E. McMahon** has been active in the software engineering field since 1973 after receiving his master's degree in mathematics from the State University of New York at Binghamton (now Binghamton University). Paul began his career as a software developer, spending the first twenty-five years working in the US Department of Defense modeling and simulation domain. Since 1997, as an independent consultant/coach (<http://pemsystems.com>), Paul helps organiza-

tions and teams using a hands-on practical approach focusing on agility and performance.

Paul has taught software engineering at Binghamton University, conducted workshops on software engineering and management, and has published more than 50 articles and 5 books. Paul is a frequent speaker at industry conferences. He is also a Senior Consulting Partner at Software Quality Center. Paul has been a leader in the SEMAT initiative since its initial meeting in Zurich.

## Michael Goedicke



**Prof. Dr. Michael Goedicke** is head of the working group Specification of Software Systems at the University of Duisburg-Essen. He is vice president of the GI (German National Association for Computer Science), chair of the Technical Assembly of the IFIP (International Federation for Information Processing), and longtime member and steering committee chair of the IEEE/ACM conference series Automated Software Engineering. His research interests include, among others, software engineering methods, technical specification and realization of software systems, and software architecture and modeling.

He is also known for his work in views and viewpoints in software engineering and has quite a track record in software architecture. He has been involved in SEMAT activities nearly from the start, and assisted in the standardization process of Essence—especially the language track.