4 Identifying the Key Elements of Software Engineering

The goal of this chapter is to present the key elements of the development endeavor, which later become "alphas," the building blocks of Essence—the things we work with when developing software. In this chapter, we discuss

- the key elements within software engineering that deliver value to the customer;
- the key elements in software engineering that are related to the targeted solution and development endeavor; and
- the role and importance of different stakeholders, requirements, and team composition.

This knowledge will help us to lay out a model of software engineering with areas of concern and key elements, which will create the basis for our understanding of Essence. To understand this model in practical application, we now rejoin Smith in his journey into software development.

4.1 Getting to the Basics

After Smith had been working in the software industry for several years, he had his fair share of ups and downs. He wished there were more ups than downs. Without a doubt, software engineering is a creative process, but Smith had come to recognize that there are some fundamental basics—some things to be mindful of, to avoid making unnecessary mistakes.

Smith's colleagues also recognized that, but they had difficulty articulating these fundamentals due to their different backgrounds, experience, and, consequently, the different terms they used. It seemed that every time a new team was formed, members had to go through a "storming and norming" process to iron out these terms before starting to deal with the challenges.

If you have been in the software industry for some time, you can empathize with Smith. For students new to software engineering, we want you to appreciate the complexities of a software development endeavor as you read this chapter and compare that against the complexities of your class, or of project assignments that you have worked on.

Essence was developed to help people like Smith and companies like Travel-Essence who rely heavily on software to run their business. What the contributors to Essence did was to lay down the foundation of software engineering for folks like Smith and yourself to cut short this startup period and ensure health and speed as your software development endeavors progress. The term health is discussed and defined later on for the area of software development. See, for example, Chapter 11 for a more detailed discussion.

Let's begin with some commonly used terms found in software engineering, which we will briefly introduce in *italics*. Regardless of size and complexity, all software development endeavors involve the following facets (see Figure 4.1):

- There are *customers* with needs to be met.
    - Someone has a problem or *opportunity* to address.
    - There are *stakeholders* who use and/or benefit from the solution produced, and some of these will fund the endeavor.
- There is a *solution* to be delivered.
    - There are certain *requirements* to be met.
    - A *software system* of one form or another will be developed.
- There is an *endeavor* to be undertaken.
    - The *work* must be initiated.
    - An empowered *team* of competent people must be formed, with an appropriate *way of working*.

These terms map out what software engineering is about. When something goes wrong, it is normally an issue with one or more of these facets. The way we handle these issues has a direct impact on the success or failure of the endeavor. We will

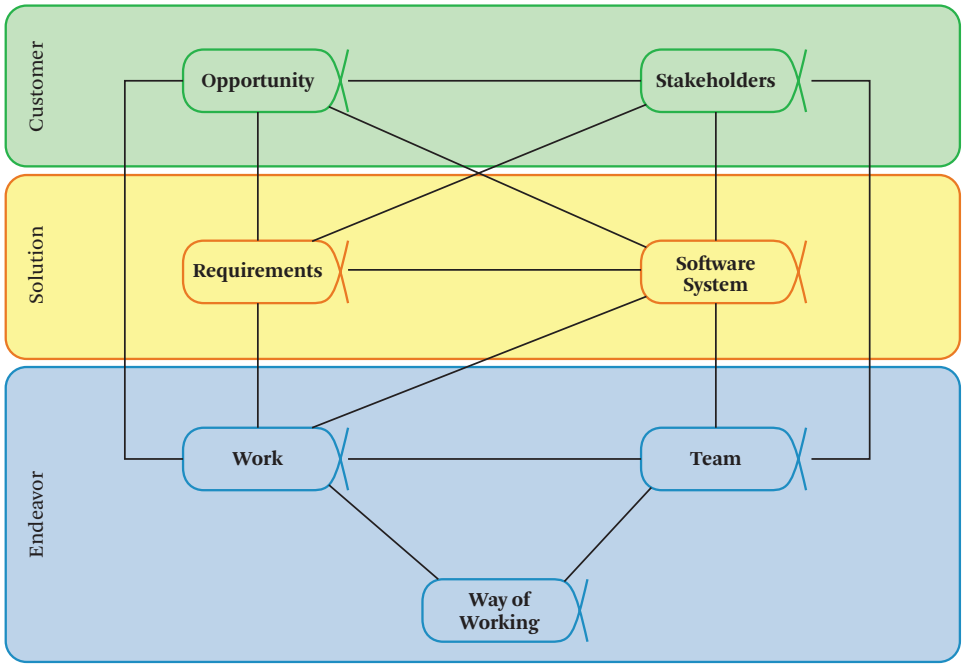


Figure 4.1 The things involved in all development endeavors.

now look at each of these facets in turn. Later, in Chapter 6, we will once more discuss issues and their relationships.

4.2 Software Engineering Is about Delivering Value to Customers

First, software engineering is about delivering value to customers. This value can be in improvements to existing capabilities or in providing new capabilities that are needed by the customer. (In our TravelEssence mode, customers are the users. They can be travelers or travel agents who make reservations on behalf of actual travelers.) Different customers would have different needs. If the endeavor does not deliver value, then it is a waste of time and money. As the saying goes, life is too short to build something that nobody wants!

4.2.1 Opportunity

Every endeavor is an opportunity for success or failure. It is therefore very important to understand what the opportunity is about. Perhaps you have heard of Airbnb. Airbnb started out in 2008 with two men, Brian Chesky and Joe Gebbia, who were

struggling to pay rent. They came up with the idea of renting out three airbeds on their living-room floor and providing their guests with breakfast. It turned out that during that time, there was an event going on in their city and many participants weren't able to book accommodations. Brian and Joe realized they were onto something. To cut the story short, Airbnb became a 1.3 billion USD business in 2016.

However, not all businesses grow and are successful like that. In fact, far more companies do not make it, and miss many opportunities. In fact, there has been a 90% failure rate for startups.[1] Many successful companies become failures due to missed opportunities. Thus, understanding opportunities is critical.

An opportunity is a chance to do something, including fixing an existing problem. In our context, an opportunity involves building or enhancing software to meet a need. Regardless of what the opportunity is, it can either succeed or fail. It can deliver real value, or it could be something that nobody wants.

As an example, our TravelEssence model revealed that customers like to engage travel staff because of the recommendations that the staff provides. The opportunity here is that if TravelEssence can provide recommendations online through a software solution, it can provide better service to customers, thereby shortening the time customers need to make a purchase decision. Of course, whether this opportunity is truly viable depends on many factors.

Thus, when working with an opportunity it is important to continuously evaluate the viability of the opportunity as it gets implemented.

- When the opportunity is first conceived, some due diligence is necessary to determine if it truly addresses a real need or a real new idea that customers are willing to pay money for.
- It would likely be the case that different solution options are available to address the opportunity, and some difficult choices will have to be made.
- When the solution goes live, it normally takes some time before the benefits become visible to customers.

4.2.2 Stakeholders

For opportunities to be taken up, there must be some people involved in the decision. The name we have for those individuals, organizations, or groups is *stakeholders*. Stakeholders who have some interest or concern in the system being developed are known as external stakeholders; those interested in the development

1. https://www.forbes.com/sites/neilpatel/2015/01/16/90-of-startups-will-fail-heres-what-you-need-to-know-about-the-10/#43f76e846679

endeavor itself are called internal stakeholders. In our TravelEssence case, internal stakeholders include a development team assembled to develop the new services for travelers, along with key managers in the organization who need to agree to the new venture. Examples of external stakeholders being affected by the system include a manager in our TravelEssence organization who needs to agree to fund the new software effort, or a traveler who might benefit by using the new services.

One of the biggest challenges in a development endeavor is getting stakeholders to agree. Before that can occur, they must first be involved in some way. The worst thing that could happen is that when all is said and done, someone says, "This is not what we want." This happens too often.

Thus, it is really important early in the endeavor to:

- understand who the stakeholders are and what their concerns are;
- ensure that they are adequately represented and involved in the process; and
- ensure that they are satisfied with the evolving solution.

4.3 Software Engineering Delivers Value through a Solution

What sets a software development endeavor apart from other endeavors is that the *solution* that addresses the opportunity is via a good piece of software. Nobody wants a poor-quality product. Customers' needs are ever evolving, so the solution needs to evolve as well, and for that to happen it has to be extensible. This extensibility applies to both the *requirements* of the solution and the *software system* that realizes the requirements.

In TravelEssence, the requirements for the solution cover different usage scenarios for different kinds of customers (e.g., new travelers, frequent travelers, corporate travelers, agents, etc.). The software system involves a mix of mobile applications and a cloud-based backend.

4.3.1 Requirements

Requirements provide the stakeholders' view of what they expect the software system to provide. They indicate what the software system must do, but do not explicitly express how it must be done. They identify the value of the system in respect to the opportunity and indicate how the opportunity will be pursued via the creation of the system.

As such, requirements serve like a medium between the opportunity and the software system. Among the biggest challenges software teams face are changing requirements. Usually, there is more than one stakeholder in an endeavor, and stakeholders will of course have different and even conflicting preferences and

opinions. Even if there is only one stakeholder, he/she might have different opinions at different times. Moreover, the software system will evolve together with the requirements. What we see affects what we want. Thus, requirement changes are inevitable because the team's understanding of what's needed will evolve. What we want to prevent is unnecessary miscommunication and misunderstanding.

At TravelEssence, Smith encountered this when working on a discount program. The team had thought that this enhancement would be very simple. However, the stakeholders had different ideas on how long the program should last, which group of users the discount program should focus on, the impact of the discount program on reservation rates, etc. They had wanted to launch the discount program within a month's time, yet there was a great deal of debate even to the very last hour.

Thus, how a team works with requirements is absolutely crucial, with principles like:

- ensuring that requirements are continuously anchored to the opportunity;
- organizing the requirements in a way that facilitates understanding and resolves conflicting requirements;
- ensuring that the requirements are testable, i.e., that one can verify that the software system does indeed fulfill the requirements without ambiguity; and
- using the requirements to drive the development of the software system. In fact, code needs to be well structured and easy to relate back to the requirements. Progress is measured in terms of how many of the requirements have been completed.

4.3.2 Software System

The primary outcome of a software endeavor is of course the software system itself. This software system can come in one of many different forms. It could be the app running on your mobile phone; it could be embedded in your air conditioner; it could help you register for your undergraduate program; it could tally election votes. It could run on a single machine or be distributed on a server farm in data centers or across a vast network as in the Internet today.

Whatever the case, there are three important characteristics of software systems necessary before they can be of value to users and stakeholders: functionality, quality, and extensibility.

The need to have *functionality* is obvious. Software systems are designed and built to make the lives of humans easier. They each must serve some functions, which are derived from the software system's requirements.

The need for *quality* is easy to understand. Nobody likes a software system that is of poor quality. You do not want your word processor to crash when you are finishing your report, especially if you have not saved your work. You want your social media posts to be instantaneous. Thus, quality attributes like reliability and performance are important. Other qualities, such as ease of use or a rich user experience, are becoming more important as software systems become more ubiquitous. Of course, the extent of quality needed depends on the context itself. This again can be derived from the software system's requirements.

The third characteristic is that of being *extensible.* It can be said that this is another aspect of quality, but we want to call this out separately. Software engineering is about changing and evolving the software system, from one version to the next, giving it more and more functionality to serve its users. This evolution occurs over time as a series of increments of more functionality, where every increment is described by more requirements. This is illustrated in Smith's job at TravelEssence, which involves introducing changes to the existing travel reservation software system when TravelEssence introduces new discount programs, initiates membership subscription incentives, integrates with new accommodation providers, etc.

There are several important aspects of this evolution. First, it does not merely entail hacking or patching code into the software system. Otherwise, as the software system grows in size, it will be harder to add new functionality. Consequently, teams often organize software systems into interconnecting parts known as components. Each component realizes part of the requirements and has a well-defined scope and interface. As a student, the lessons you will learn about object orientation, etc. are about organizing the software system into manageable components.

Second, code needs to be well structured and easy to relate back to the requirements. Just as the requirements will evolve, the software system needs to be extensible to such changes.

Third, the evolution involves transitioning the software system across different environments, from the developer's machines, to some test environment, to what is known as the production environment, where actual users will be using the software system. It is not unusual to find that software that works on the developer's machines will have defects (or bugs) in the test or production environment. Many senior developers get irritated when they hear novices say: "But it works on my machine." A developer's job is not done until they system works well in the production environment. A quality software system must:

- have a design that is a solution to the problem and agreed to;
- have demonstrated critical interfaces;

- be usable, adding value to stakeholders; and
- have operational support in place.

4.4 Software Engineering Is Also about Endeavors

An endeavor is any action that we take with the purpose of achieving an objective, which in our case means both delivering value according to the given opportunity and satisfying stakeholders. Within software engineering, an endeavor involves a conscious and concerted effort of developing software from the beginning to the end. It is a purposeful activity conducted by a software team that achieves its goals by doing work according to a particular way of working.

4.4.1 Team

Software engineering involves the application of many diverse competencies and skills in a manner similar to a sports team. As such, a team typically involves several people and has a profound effect upon any development endeavor. Without the team there will be no software.

Good teamwork is essential for high performance. It creates a synergy, where the combined effect of the team is much greater than the sum of individual efforts. But getting to a high-performance state does not come naturally; instead, it results from a deliberate attempt to succeed.

To obtain this high level of performance, the team members should reflect on the way they work together and how they focus on the team goal.

Teams need to:

- be formed with enough people to start the work;
- be composed of personnel possessing the right competencies/skills;
- work together in a collaborative way; and
- continuously adapt to their changing environment.

When working in TravelEssence, Smith belonged to a development team. Although members of Smith's team had slightly different skill sets, they collaborated to achieve the team's goal together. Smith was particularly focused on backend technologies (i.e., the part of the software system running on the cloud), whereas Grace, a colleague, focused on front-end JavaScript and React Native technologies. (Since these technologies are not the emphasis of this book, you don't need to know them. Moreover, technologies change rather quickly. Instead, what we want to

emphasize is that effective teams have to address the opportunity presented to them to satisfy stakeholders.)

4.4.2 Work

When a team comes together to do the work of making the opportunity a reality in the software system they build, the purpose of all their efforts is to achieve a particular goal. In general, there is a limited amount of time to reach this goal: they must get things done fast but with high quality. The team members must be able to prepare, coordinate, track, and complete (stop) their work. Success in this has a profound effect upon meeting commitments and delivering value to stakeholders. Thus, the team members must understand how to perform their work and recognize if the work is progressing in a satisfactory manner.

Doing the work, then, involves:

- getting prepared;
- communicating the work to be done;
- ensuring that progress and risk are under control; and
- providing closure to the work.

In TravelEssence, Smith and his team managed their work through a backlog. The backlog is a list of things to do, which originated from requirements. They communicated regularly with their stakeholders to make sure that their backlog was accurate, up-to-date, and represented the stakeholders' priorities. In this way, they could focus on getting the right things done.

4.4.3 Way of Working

A team can perform their work in different ways and this may lead to different results. It can be performed in an ad hoc manner, meaning that you decide how to work while doing the work. For instance, while cooking a soup, you may not follow a recipe—you might decide on the fly which ingredients to use and in what order to mix and cook them together. When following an ad hoc way of working, the result may or may not be of high quality. This depends on many factors: among them, the skill of the people involved and the number of people involved in the process.

All of us are acquainted with the saying "too many cooks spoil the broth." If too many cooks cook the soup in an ad hoc manner, the soup won't taste good. Translating the analogy to software, if too many people participate in agreeing on how to do the job, the job will probably not be done well. There are many reasons

for this. One of them is that each person has his/her own idea of how to conduct the job and, often, they do not work in an orchestrated manner.

Smith's team addressed this issue by regularly looking at their prioritized backlog. They made sure that they correctly understood the scope of each item in the backlog, checking with stakeholders and getting feedback from them. Smith's team regularly examined their method or, in other words, their way of working. If things did not seem right, they made changes.

It is therefore important for team members to come into some kind of consensus regarding their way of working. Disagreements about the way of working are significant barriers to team performance. You would think that coming to an agreement would be easy. The truth of the matter is that it is not. On a small scale, within a single team, there is still a need for members to agree on the foundations and principles, followed by specific practices and tools. This would of course need to be adapted to the team's context, and evolve as the environment and needs change.

The way of working must:

- include a foundation of key practices and tools;
- be used by all the team members; and
- be improved by the team when needed.

In an industry scale, one of the things we hope to achieve through Essence is to simplify the process of reaching a common agreement. We do that at this scale by identifying a common ground or a kernel and having a way to deal with diversity of approaches. In the subsequent chapters, we will discuss the approach taken by Essence in greater detail.

In this chapter, we have introduced the following terms: opportunity, stakeholders, requirements, software system, work, team, and way of working. Essence will give these terms greater rigor and provide guidance to software teams on how to build a stronger foundation to achieve their goals.

What Should You Now Be Able to Accomplish?

After studying this chapter, you should be able to:

- list and explain the things involved in all development endeavors, related to the customer (i.e., opportunity, stakeholders), solution (i.e., requirements, software system), and endeavor (i.e., work, team, way of working);
- give examples of different types of stakeholders, together with their interests and concerns;

- explain the mediating role of requirements;
- name and explain the three key characteristics of software systems (i.e., functionality, quality, and extensibility); and
- explain what makes a good team.

Understanding the facets of software engineering covered in this chapter provides an overview of the main core of Essence. This core may seem fairly abstract at this point, but as you read on, you will recognize all these facets in the Essence alphas, and be able to apply them in more and more practical and detailed ways.

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. 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, 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 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. 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. 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. 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. 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. 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. 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. DOI: 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. 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

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.

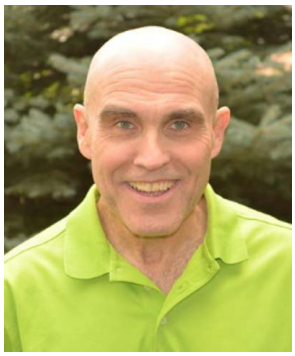
Tragically, Harold Lawson passed away after battling an illness for almost a year, just weeks before the publication of this book.

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 organizations 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.