

Running with Scrum

The goal of this chapter is to introduce the Scrum practice, including its elements in the Essence language, and use the TravelEssence example to demonstrate the benefits of Scrum compared to the development approach used earlier in the book. In this chapter, the reader will be introduced to

- the Scrum practice and its elements;
- the relationships between the Scrum elements, activity flows, and to their relationships with kernel elements (i.e., Work, Requirements, Software System);
- the simplified version of the Scrum practice (called Scrum Lite) in a real endeavor, including the obstacles and challenges that might arise; and
- how kernel activity spaces are covered by the Scrum Lite practice.

As mentioned previously, Cheryl (the CIO) had mandated after a series of successful endeavors, that Scrum and either user stories or use cases be employed by all development teams. One reason why organizations often mandate specific practices and tools is to simplify training and communication. Recall that a practice is defined to be *a repeatable approach to doing something with a specific purpose in mind*. By making a practice explicit we improve communication, reducing the chances of someone misunderstanding how the practice is intended to be carried out.

Scrum is perhaps the most popular agile practice at the time of this writing. Jeff Sutherland and Ken Schwaber created Scrum to get teams to work iteratively and to collaborate more effectively by following a number of practical and proven activities.

14.1 Scrum Explained

Figure 14.1 shows a big picture overview of Scrum [Schwaber and Sutherland 2016]. It provides explicit guidance related to how a small development team with about

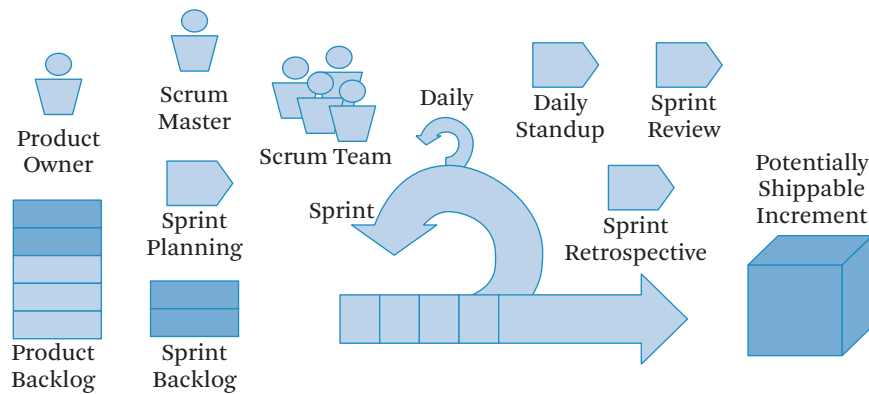


Figure 14.1 Scrum big picture.

7 plus/minus 2 team members can work together. Henceforth, we will refer to that team as a Scrum team. All the work that the team might have to do is first placed in an ordered list—the product backlog. The product backlog is maintained with the most important items near the top of the list. The things in the product backlog are called *Product Backlog Items* (PBIs). A PBI can be a piece of a requirement, something the team can do to improve themselves, or defects that they will have to fix.

The heart of Scrum is the sprint, a fixed-length period of time, usually one to four weeks, during which the team meets a certain goal, which includes producing a potentially shippable increment of the product to be developed. The PBIs to be performed in a sprint are selected through the sprint planning activity where the team, together with the product owner (PO), agrees on the highest prioritized PBIs to be worked on in the upcoming sprint. These PBIs are moved from the product backlog to the sprint backlog. This activity is done on the first day of each sprint by the full Scrum team working together to determine what can be delivered and how it can be delivered in the agreed-to sprint time period. There are two parts to the sprint planning activity. During the first part, the PO explains to the team the goals of the sprint and the Product Backlog Items that, if implemented, would achieve the sprint goal.

The goal gives the Scrum team some flexibility regarding the functionality implemented within the sprint. During the second part of the meeting, the team forecasts the PBIs it will achieve and determines their agreed-upon sprint goal. Those PBIs are then moved to the sprint backlog (i.e., the agreed work to be done in the current sprint).

Each day during the sprint, the team meets to synchronize their work and create a plan for the next 24 hours. This is called the daily scrum (or daily standup, as shown in Figure 14.1) and is limited to 15 minutes. At the daily scrum, each team member explains what he/she did since the last meeting, what s/he plans to do today, and what is getting in her/his way preventing her/him from meeting the sprint goal. Solutions to problems are not discussed in the daily scrum. A separate meeting is arranged to dive deeper into problems when necessary.

At the end of the sprint, the team conducts a sprint review activity with key stakeholders to review the product in its current version (referred to as a potentially shippable increment) they have produced. At this review, stakeholders may also identify product improvements (PBIs) that will be placed in the product backlog. At the end of each sprint, the team holds a sprint retrospective activity. The sprint retrospective is an opportunity for the Scrum team to agree on improvement to their way of working, to be implemented in the next sprint.

There are three major roles in Scrum, namely the PO, the Scrum master, and developers. The PO is responsible for feeding the product backlog based on his/her interaction with customers and users. The PO is also responsible for prioritizing the PBIs. The team members (i.e., developers) are responsible for estimating the effort for implementing each PBI.

The Scrum master role is something unique to Scrum. The Scrum master is a servant leader, a person who facilitates the Scrum activities and motivates the team members to follow the Scrum activities.

While the Scrum activities we have just described are fairly simple, teams often tailor them based on their own situations. This is one reason why capturing a team's agreed-to way of working as a set of explicit practices can help team members—especially new and less experienced team members—understand what activities are expected, what options they have in carrying out these activities, and how much detail is expected in any related work products.

14.2 Practices Make a Software Engineering Approach Explicit and Modular

Scrum is in essence a practice, or rather, a set of practices. Briefly speaking, a practice is about doing stuff in a certain way to address certain problems, and with Scrum, it is about teams improving team collaborations and performance. We will take a slight detour to explain how Essence captures practices in an explicit way and thereby provides practical guidance to teams. We will in a short moment capture the Essence of Scrum and demonstrate how Smith's team applied Scrum.

The word “practice” is an overloaded word, meaning different things to different people. The Essence specification provides a specific definition: *a practice is a repeatable approach to doing something with a specific purpose in mind*. A practice provides a systematic and verifiable way of addressing a particular aspect of the *work* at hand. It has a clear goal expressed in terms of the results its application will achieve. It provides guidance not only to instruct practitioners in what is to be done to achieve the goal, but also to ensure that the goal is understood and to verify that it has been achieved.

As such, a practice provides a proven way of approaching or addressing a problem. It is something that has been done before, can be successfully communicated to others, and can be applied repeatedly, producing consistent results.

14.3 Making Scrum Explicit Using Essence

Scrum can be represented as a practice that is a set of activities to help teams conduct iterative development in a highly collaborative manner. In Part [IV](#) of this book, we will show a different way to represent Scrum as a composition of multiple practices: that is, a product ownership practice, a backlog practice, an iterative development practice, and a retrospective practice. Thus, Scrum is not simple, and indeed, the Scrum Guide [[Schwaber and Sutherland 2016](#)] calls Scrum a process framework. For the purpose of this part of the book, we choose to use a simplified version of Scrum, which we represent as a single practice that we call Scrum Lite.¹ While our Scrum Lite includes what we have assessed to be the important elements of Scrum, we do not include a discussion on the ideas of Scrum or all of the responsibilities of all of the Scrum roles, nor do we include all of the characteristics of a Scrum Team.

Different practice authors will have different ways to express their key concepts. So, Scrum will be expressed in one way (as depicted in [Figure 14.1](#)), user stories another way, use cases another way, and microservices yet another way. These authors haven’t used a common ground, because they haven’t had one. They use different terms for the same thing, and the same term may have different meaning in their practices. Some authors deal with this problem by re-describing what the original author developed, but they don’t use a standard like Essence; rather, they create their own terms. This doesn’t create a collaborative environment between the people who have contributed their ideas to the world. This is one of the serious problems Essence addresses—under Essence, every author uses the same standard

1. Based on version 03.2015 of the Scrum Essentials practice originally developed by Ivar Jacobson International. Used and adapted with permission.

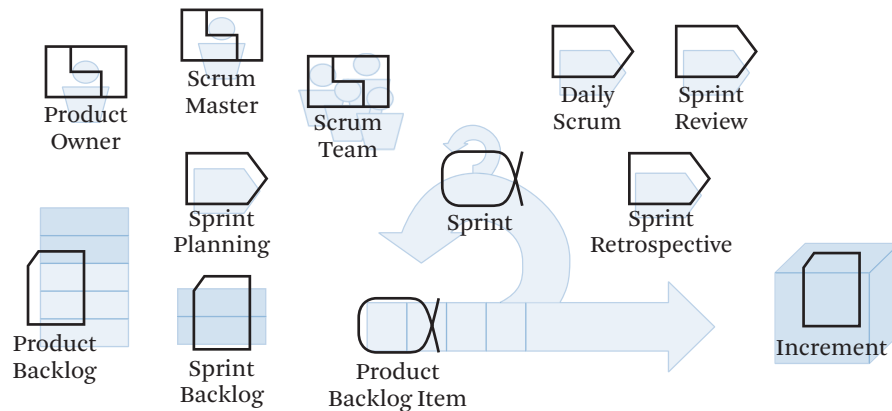


Figure 14.2 Scrum big picture mapped to the Essence language.

language. It becomes significantly easier for students to learn many practices. Once they have learned one practice, they will while learning a new practice recognize that there is a lot in common, even if the new practice covers a different topic than the first practice. And the more practices they learn, the easier it becomes to learn something new.

We can redraw the Scrum Lite big picture in Figure 14.1 using the Essence language, as shown in Figure 14.2. So, what you see in this figure is the Scrum big-picture elements represented using the Essence language symbols. To start, the figure reminds us that PO, Scrum Master, and Scrum Team roles are represented as patterns in the Essence language. And Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective are activities. Product Backlog, Sprint Backlog, and Increment are work products. And finally, Sprint and Product Backlog Item are alphas in the Essence language.

A complete model of the Scrum practice is shown in Figure 14.3. It is a useful diagram in that it shows

1. relationships between the elements in the practice, such as the relationships between Sprint, Sprint Backlog, Product Backlog, PBI, and Increment;
2. activity flows, such as from Sprint Planning to Daily Scrum to Sprint Review and Sprint Retrospectives; and
3. relationships with kernel elements, such as between Sprint and Work, PBI and Requirements, and Increment and Software System.

Thus, Figure 14.3 not only shows the relationships between elements in the practice, in this case the Scrum practice, but also the relationship with the kernel, and

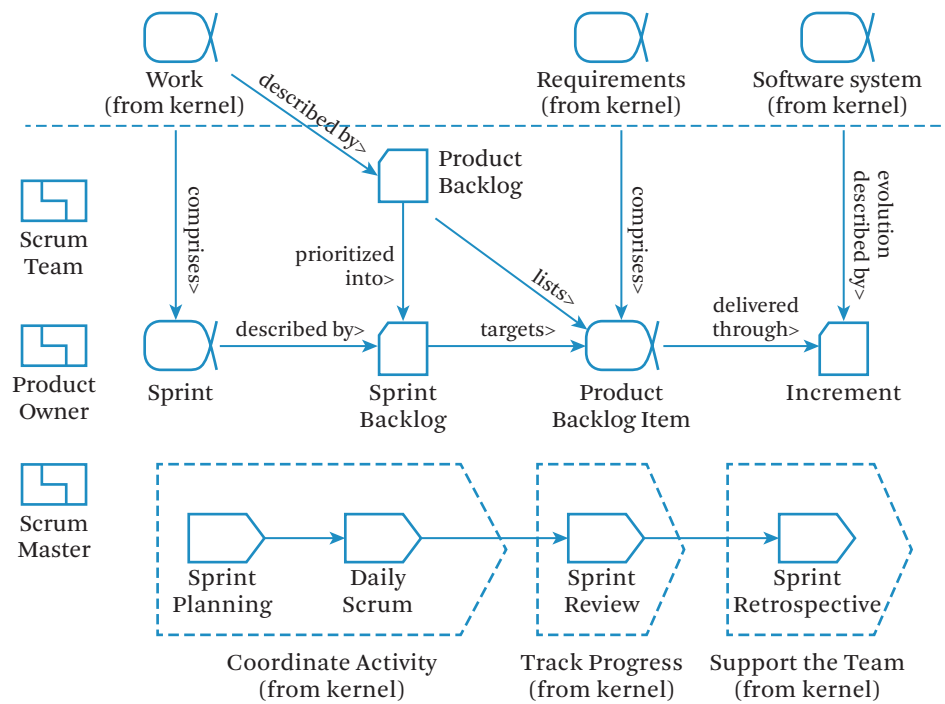


Figure 14.3 Scrum Lite practice expressed in the Essence language.

through that you are better able to understand how Scrum provides more guidance on top of what is available in the kernel.

An important concept highlighted in Scrum is the “Definition of Done” (DoD). The DoD is a clear and concise list of criteria that each PBI must satisfy for the team to call it complete. As an example, a PBI may include DoD criteria such as:

- sufficiently tested;
- accepted by the PO;
- source code checked in; and
- associated documents (e.g., user manuals) updated.

The DoD must apply to all items in the backlog. It can be considered a contract between the Scrum team and the PO. The purpose of DoD is similar to the purpose of the checklists of alpha states. They provide an explicit definition of what the team members must do. DoD as provided through the alpha state checklists is available

for PBI and Increments, and in general any alpha or work product. Alpha states go one step further by defining the completion criteria for each state.

Table 14.1 provides a summary of the elements in the Scrum Lite practice. Before we look in on the TravelEssence team's use of them, we will establish the framework for these elements.

Table 14.1 Elements of Scrum Lite practice

| Element | Type | Description |
|----------------------|--------------|---|
| Sprint | Alpha | A time-box (e.g., fixed length of time) of one month or less during which a “Done,” usable and potentially shippable product increment is created. A new sprint starts immediately after the conclusion of the previous sprint. |
| Product Backlog Item | Alpha | A change to be made to the product (in a future release, for example, a feature, user story, requirement, enhancement, or fix). |
| Sprint Backlog | Work product | The set of PBIs selected for the sprint, plus a plan for delivering the Increment and realizing the sprint goal. The sprint backlog makes visible all of the work the development team identifies as necessary to fulfill the sprint goal. |
| Product Backlog | Work product | A priority-ordered list of everything that might be needed in the product: the single source of requirements for any changes to be made to the product. The items in the product backlog are known as PBIs. |
| Increment | Work product | The sum of all PBIs completed during a sprint and those items completed during all previous sprints. The increment must be “Done,” which means the software system it describes must be usable and meet the Definition of Done (DoD). When a PBI or an increment is described as “Done,” everyone must understand what “Done” means. Although this varies significantly per Scrum team, members must have a shared understanding of what it means for work to be complete, to ensure transparency. This is the definition of done for the Scrum team and is used to assess when work is complete on the product Increment. (Work completed includes only items that meet the team's agreed-to DoD.) |

Table 14.1 (continued)

| Element | Type | Description |
|----------------------|----------|---|
| Daily Scrum | Activity | The team meets every day, same time and place, to assess progress, synchronize activity, and raise any issues that are getting in their way and require action to resolve. The meeting is time-boxed, typically to 15 minutes. |
| Sprint Planning | Activity | Deciding what can be delivered in the sprint's increment and how the work needed to deliver the agreement will be achieved. |
| Sprint Review | Activity | A time-boxed review of the outcomes of the sprint, used to gather feedback and discuss what should be done next. |
| Sprint Retrospective | Activity | The whole team meets at the end of the sprint to reflect on its way of working. Improvements are identified and prioritized, and actions agreed. At the next retrospective, the results are evaluated. |
| Product Owner | Pattern | <p>The PO is responsible for maximizing the value of the product and the work of the development team. How this is done may vary widely across organizations, Scrum teams, and individuals. The PO is the sole person responsible for managing the product backlog.</p> <p>Product backlog management includes clearly expressing PBIs:</p> <ul style="list-style-type: none"> • ordering the items in the product backlog to best achieve goals and missions; • optimizing (maximizing) the value of the work the development team performs; • ensuring that the product backlog is visible, transparent, and clear to all, and shows what the Scrum team will work on next; and • ensuring the development team understands items in the product backlog to the level needed. |
| Scrum Master | Pattern | The Scrum master is responsible for ensuring that Scrum is understood and enacted. The Scrum master is a servant leader for the Scrum team. Among other things, the Scrum master helps to: |

Table 14.1 (continued)

| Element | Type | Description |
|------------|---------|---|
| | | <ul style="list-style-type: none"> • guide Scrum activities; • remove impediments; • ensure everyone understands Scrum; and • make certain all members of the Scrum team understand the need for clear and concise product backlog items. |
| Scrum Team | Pattern | The Scrum team consists of a PO, the development team, and a Scrum master. Scrum teams deliver products iteratively and incrementally, maximizing opportunities for feedback on how they are doing and self-improvement. The best Scrum team size is small enough to remain nimble and large enough to complete all significant work within a sprint. |

14.4 Scrum Lite Alphas

In this section, we will introduce each of the alphas that comprise the Scrum Lite practice. These alphas are the Sprint alpha and the PBI alpha.

14.4.1 Sprint

As mentioned, a sprint is a time-box (e.g., fixed length of time) whereby some useful work is completed.

In the Scrum guide [Schwaber and Sutherland 2016], there are no explicit alpha states defined. However, we find that alpha states are very useful for teams who are new to Scrum. They help teams understand what they must do to prepare for their sprints and for each activity in a sprint. In our Scrum Lite practice the Sprint alpha states are as follows (see also Figure 14.4 below).

Scheduled. The start and end dates for the sprint are scheduled and all team members are aware of the dates; there are sufficient backlog items in the product backlog, and the backlog items have been prioritized.

Planned. The goals to be achieved in the sprint have been agreed, including the specific backlog items within the scope of the Sprint, and the risks have been identified and the team has agreed how to mitigate them.

Reviewed. The outcome of the sprint has been reviewed. This includes reviewing the product increment (such as the recommendation engine), and the team's way of working. This includes reviewing which specific backlog items

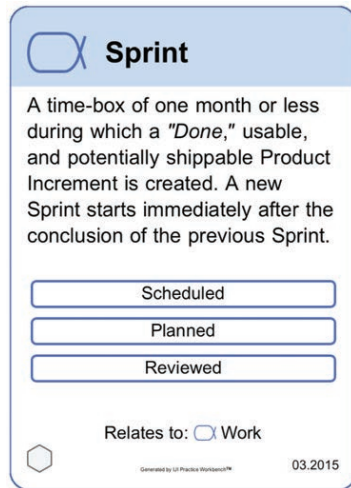


Figure 14.4 The Sprint alpha card.

have been completed, which ones were not completed, why they were not completed, how the team can improve their way of working, and what improvement actions they can take.

In companies, these Sprint alpha states will often be tacit. However, for students new to Scrum, these states and checklists are very useful; new students have not learned the essence of Scrum and can't easily figure it out on their own. By providing the checklists, a less experienced developer would be guided away from making unnecessary mistakes. Figure 14.5 shows the alpha state checklists.

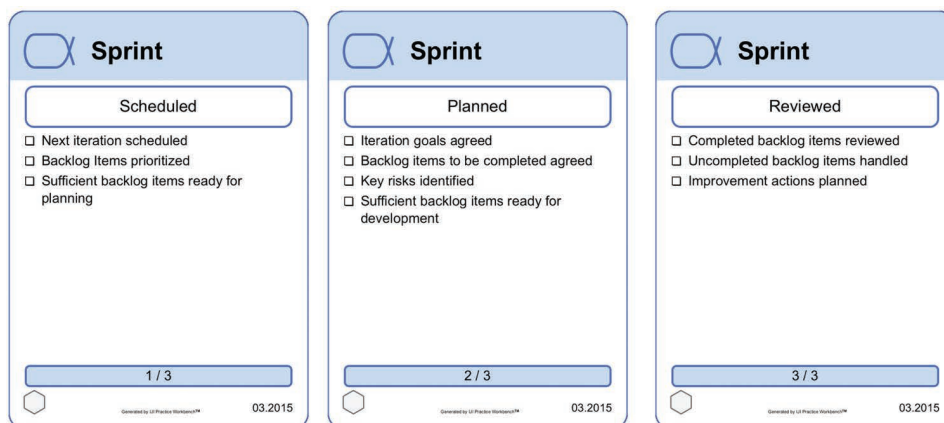
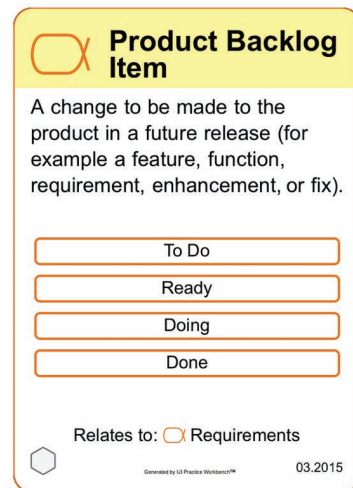


Figure 14.5 The Sprint alpha state cards.



Product Backlog Item

A change to be made to the product in a future release (for example a feature, function, requirement, enhancement, or fix).

To Do

Ready

Doing

Done

Relates to: Requirements

Generated by UI Practice Worksheets™ 03.2015

Figure 14.6 The Product Backlog Item alpha card.

These simple checklists are also a good source of information for learning and as reminders even for experienced developers, who can still manage to forget what they previously learned. However, being too prescriptive might prevent the practitioner from thinking on his/her own, and instead treating the checklists as the word of God. Like all things, this is about striking a balance between what should be left tacit and what should be made explicit. In our Scrum Lite practice, we intentionally did not add a fourth state, for Retrospective Completed, to the Sprint alpha. The value of such a state is something each organization can decide based on its own situation, including the competencies of its people.

14.4.2 Product Backlog Item

A Product Backlog Item, or PBI, is a change to be made to the product in a future release (for example, a feature, user story, requirement, enhancement, or fix).

The alpha has the following states identified in Figure 14.6.

To Do. It has been agreed that the PBI needs to be completed within the next sprint. The scope and completion criteria of the PBI are clear.

Ready. The team works together with the product owner to agree on how they should go about completing the PBI.

Doing. At this state, the team is working on the item and bringing it to completion.

Done. The Product Backlog Item has been completed.

14.5 Scrum Lite Work Products

The Scrum Lite practice comprises the following work products:

- Product Backlog,
- Sprint Backlog, and
- Increment.

14.5.1 Product Backlog

A product backlog work product is a priority-ordered list of everything that might be needed in a product. It is the single source of requirements for any changes to be made to the product (see Figure 14.7). The items in the product backlog are known as PBIs.

For endeavors at this point, there is only one level of detail in a product backlog:

Items Ordered. The product backlog Items are captured in the product backlog, which can be in the form of a spreadsheet or within some backlog management tool. They are ordered according to their priority, so that high priority ones can be selected for the next sprint backlog.

For more sophisticated endeavors, there might be more levels of detail. For example, the team might want to describe rationales for prioritizing the PBIs, so that team members can avoid unnecessary debates.

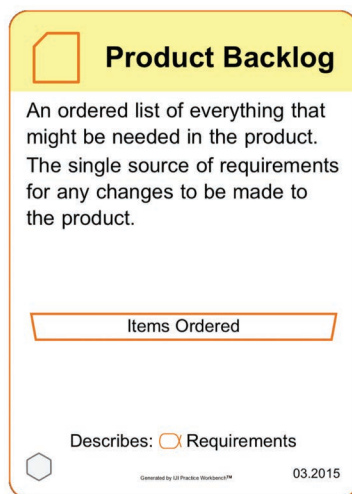


Figure 14.7 The Product Backlog work product card.

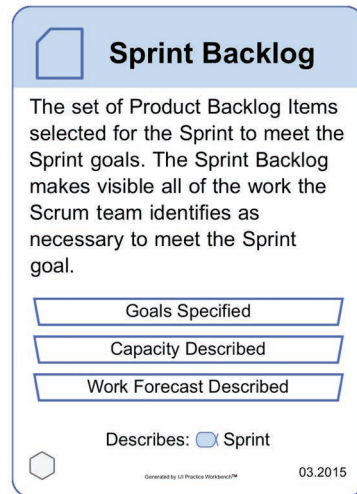


Figure 14.8 The Sprint Backlog work product card.

14.5.2 Sprint Backlog

A sprint backlog work product is the set of PBIs selected for a sprint. It also includes a plan for delivering an increment realizing the agreed-upon sprint goal. A sprint backlog makes visible all of the work the development team identifies as necessary to meet the sprint goal.

The Sprint Backlog comprises the following levels of detail (see also Figure 14.8).

Goals Specified. The sprint goal is clearly stated and sets the target for the team members.

Capacity Described. The amount of work the team can perform is estimated. In this way, the team can determine if it has too much or too little work in the sprint.

Work Forecast Described. The team agrees on product backlog items that can be completed within the sprint, as well as target dates they expect to complete within the sprint.

14.5.3 Increment

An increment is the sum of all product backlog items completed during a sprint and those items completed during all previous sprints.

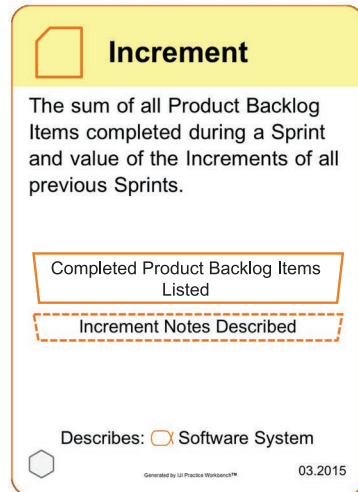


Figure 14.9 The Increment work product card.

The Increment work product has the following levels of detail (see also Figure 14.9).

Completed PBIs Listed. The PBIs that make up the Increment are clearly listed.

Increment Notes Described. Further information about the Increment is provided, such as environments in which the increment can work, known issues, and so on. By environment we mean which browser version, which operating system, and the like. The specific content has to be agreed upon by the team.

14.6 Scrum Lite Roles

Scrum Lite explicitly identifies two roles, namely the PO and Scrum Master. A role is a list of responsibilities that one or more people accept. The individuals serving as PO and Scrum Master and the rest of the team members form the Scrum team. Essence allows you to model roles and team organization as patterns.

In Part I, you learned that Essence provides a concise representation of patterns as poker-sized cards. Figure 14.10 shows the PO pattern card, which comprises the most important information about the Product Owner's responsibilities. The card shows that the PO is responsible for managing the product backlog, ensuring each item is clear to the team members, and making certain that the product backlog is visible to the team. The card also shows that the PO is responsible to ensure

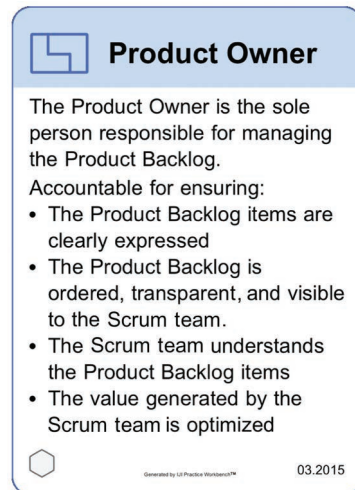


Figure 14.10 The Product Owner pattern card.

the value generated by the Scrum team is optimized, which means the work of the Scrum team provides value toward achieving the goal of the sprint.

The Scrum Master role is also represented by a pattern, as in Figure 14.11. The individual acting as the Scrum Master coaches the team as they conduct the Scrum activities. When team members face impediments, such as unclear backlog items, he/she works to remove the impediment. As an example, Smith's team faced an impediment with using a library, which required licensing fees. If the licensing were not resolved, Smith's team might have to rewrite a part of the software. Smith discussed the problem with the company legal representative and finance officer. After discussion, they agreed to use an alternative library that didn't require licensing fees, which resolved the impediment.

The third pattern in Scrum Lite is the Scrum team, which is a team pattern in Figure 14.12. The Scrum team consists of members, two of which play the roles of a PO and a Scrum Master. Scrum teams are self-organizing, which means no one outside the team tells the team how to achieve the goal of each sprint. The team includes people with the needed competencies to accomplish all the required work. This is sometimes referred to as a cross-functional team.

These cards can be used by Scrum team members by placing them on a board or having them carry them in their pockets where they can be easily accessed as quick reminders of their agreed-to responsibilities.

Thus, Angela agreed to be the PO, and Smith agreed to take on the Scrum Master role for the development team.

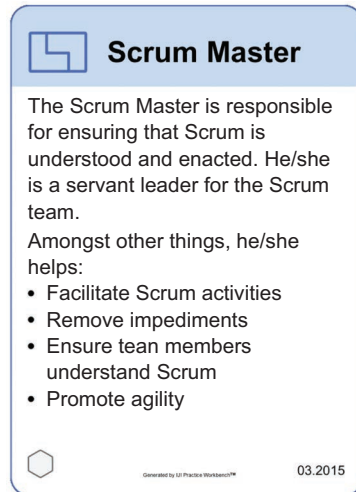


Figure 14.11 The Scrum Master pattern card.



Figure 14.12 The Scrum Team pattern card.

14.7 Kick-Starting Scrum Lite Usage

When Smith's team discussed how they would apply Scrum Lite, there were many questions. Tom, a rather senior and always vocal developer, asked, "How would using Scrum Lite differ from what we did when we delivered the demo? We have been providing demos to Angela on a weekly basis and we adapted our plans based on her feedback."

Smith replied, "Yes indeed, that is true, but Scrum Lite does have some things we can learn that can help us improve the way we are currently working. We did not do Daily Scrums to highlight problems early and improve our team communication. We did not actively manage our Requirement Items in a backlog. We did not define and assign roles with explicit responsibilities, such as Product Owner and Scrum Master. This might have been OK when we were producing the demo for Dave and Angela. But now that we are working toward a live product, and our endeavor has greater visibility to management, we need a way to ensure our team operates with appropriate discipline. We also did not really prioritize our work when we did the demo, nor did we estimate the effort." Making Scrum activities explicit helps team members apply the practices they have agreed to use. The explicit definitions act as reminders to team members. Also, by using Essence to describe practices, we will be able to see any gaps in our practices that we need to fix. This is because when we express practices using the Essence language, we can see which kernel alphas are being progressed, and we can see which ones are not being affected. If an alpha is not being affected by a certain practice, it should lead the team to ask if they have another practice that is helping them progress that alpha. They may or may not need an explicit practice for every alpha, but the team should discuss this and decide based on their specific situation. Once the team has agreed to the set of explicit practices they need, the cards can be used as visible reminders.

Tom asked, "How will the alpha state cards be used now that we are using Scrum?"

Grace replied, "I think one way would be to use the Health Monitor game to help us keep the status of each increment visible to our development team. We can easily incorporate this game into our sprint planning, review, and retrospective activities by keeping the green stickers and red stickers up to date on the board with our alpha state cards.² This can remind us to discuss problems as soon as possible."

All in all, the team members were eager to use Essence and the cards along with Scrum to help learn from their experiences. They agreed to move forward with Scrum as summarized in Table 14.2.

2. Refer to Chapter 10 for information on green and red stickers.

Table 14.2 Adopting Scrum Lite in TravelEssence

| Scrum Element | How the TravelEssence team did it |
|----------------------|--|
| Product Owner | Angela would be the PO for the team. |
| Scrum Master | Smith would be the Scrum Master for the team. |
| Sprint | Smith's team agreed that they would iterate on a weekly time-box. |
| Sprint Planning | Weekly (every Monday morning) |
| Daily Scrum | Every morning (Tuesday, Wednesday, Thursday)—note that Mondays and Fridays were for sprint planning and reviews, and for retrospectives. |
| Sprint Review | Weekly (every Friday afternoon) |
| Sprint Retrospective | Weekly (Friday afternoon after sprint review) |

14.8 Working with Scrum Lite

Smith, in his role as Scrum Master, made sure that the activities in Scrum Lite were observed in order to get the team into a working rhythm useful for nurturing good habits.

As we have outlined, working with Scrum Lite involves activities like:

- Sprint Planning,
- Daily Scrum,
- Sprint Review, and
- Sprint Retrospective.

We will further explain these activities as we see how Smith's team performs them.

14.8.1 Sprint Planning

As it does with alphas, work products, and patterns, Essence also presents the key information about activities through poker-sized cards. Figure 14.13 shows a Sprint Planning activity card.

Smith's team started using Scrum on top of Essence on one Monday morning with Sprint Planning. This was all about deciding what priority items from the Product Backlog should go into the current Sprint Backlog.

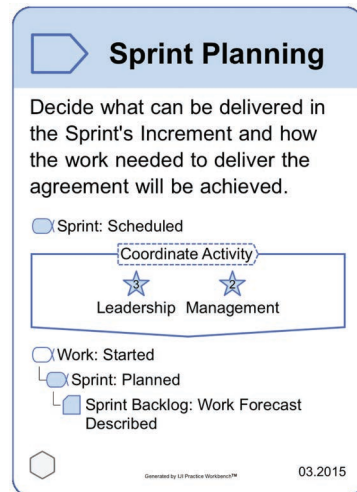


Figure 14.13 Sprint Planning activity card.

But this meant more than just picking various items from the Product Backlog and moving them to the Sprint Backlog. It was important for the team to ask some critical questions about the items they were considering.

- “Are the items we’re selecting for this sprint properly prepared?” Properly prepared means that the items are broken down into small enough sub-items to be completed by the team within the time available for the next Sprint. This means there must be enough information about each selected backlog item for the team to estimate the effort needed to complete it. You might be wondering how the team knew to ask this question. Some of the questions the team asked may come from an explicit practice defined on top of the kernel, such as the Sprint Planning activity that is part of the Scrum practice. However, other questions may be based on the kernel itself. For example, note that a Sprint is a sub-alpha to the Essence Work alpha (refer to Figure 14.4). The Work alpha contains a state named Prepared. Achieving this state means all pre-conditions for starting the work have been met. One of the checklists in this state includes the item “The work is broken down sufficiently for productive work to start.” This checklist item should remind team members to ask if the backlog items have been properly prepared, regardless of the practices they have agreed to use.

- “There are situations when such questions as above cannot be answered due to unclear/insufficient/ambiguous information in the backlog. An indication for this might be that the team is not able to reliably estimate the effort.” This meant the team needed to go back to the product owner and get more information, or reject that item as not being ready to be worked on in the sprint.
- “Has the team considered their capacity when deciding if they can commit to the proposed items to complete in this sprint?” This question comes specifically from the sprint planning activity within the Scrum practice. This means each team member needs to consider how much time they have to work on the endeavor and if they believe that is enough time to complete the items the team is committing to complete in the sprint. Note that in Figure 14.12 we saw that the Scrum team has the responsibility to be self-organizing. This means they are responsible for estimating and committing to the work to be completed within a sprint. If the team had chosen to use a different practice than Scrum, this responsibility might have fallen on a different person (for instance, a manager) in the organization. This is because another practice might define its roles differently than Scrum.

The preceding questions provide good rationale why some teams need explicit practices and some don't. Teams need to ask questions such as the following.

- “Are our team members experienced enough to know to ask key questions?” If the team does not have adequate experience, they need to decide if other team members can help, or if they need to ask for more help from outside the team. “Do they need explicit reminders, such as checklist reminders, to conduct a proper disciplined sprint planning session?” The answer to this question often depends on how many people on the team have previous experience conducting Sprint Planning. In some cases, explicit practices may be sufficient, but if most or all of the team members are new to Scrum, then a coach may also be needed to guide them through their first few Sprints. Explicit practices can provide these critical reminders to teams that have less experienced practitioners or who have never worked together before, but sometimes a coach is needed to properly set the expectations of their new team.

Both Product Backlog and Sprint Backlog have simple poker-sized cards to describe them. Figure 14.8 shows a Sprint Backlog work product card, which identifies what a team agrees to work on for a sprint. Note that this card can also provide

Table 14.3 Product Backlog

| Item | Product Backlog Item | Originator | Priority | Estimate |
|------|--|------------|----------|----------|
| 1 | Set up group of internal users | Angela | High | 2 |
| 2 | Add toggle for recommendations feature based on user group. | Angela | High | 1 |
| 3 | Run series of tests to ensure the toggle for recommendations feature does not result in performance degradation. | Angela | High | 2 |
| 4 | Add introductory user screen for recommendation functionality. | Angela | Medium | 3 |
| 5 | Develop the pre-processing data needed for the recommendation algorithm. | Smith | Low | 5 |

reminders of activities the team should be conducting while producing the work product. For example, the team needs to agree on the goals for the sprint, and it needs to estimate its capacity when it is developing its sprint backlog. In the past, often there was considerable effort put into producing lengthy documentation that wasn't used. This often occurred because practitioners tasked with producing this documentation did not have clear guidance on what should be included, along with how much detail. Work product cards provide a simple way to communicate what practitioners need to know to produce useful documentation that achieves the intent of the work product.

As mentioned earlier, the team's first target was to work toward a version of the system suitable for internal users. Angela participated as the Product Owner and she quickly identified a number of product backlog items, as shown in Table 14.3.

Since Smith's expertise and background was in the technical solutions area, he identified some key items on the product backlog, focusing on the technical issues. He said, "We need to get going on the recommendations and the user screen. These are both high-priority items." This was not meant to imply that they only needed technical issues on the backlog. As we have explained earlier, the product backlog should include all work the team has to do from the stakeholders' perspective. This will make the Product Backlog the "single source of truth" for the whole team. Everything that the team might ever need to do must eventually be added to the Product Backlog. However, at this point in the TravelEssence endeavor, the product backlog was not a complete list. It contained a number of items the team knew they

needed to do, but certainly not everything needed to create a new release. This is the way many endeavors get started.

The product backlog evolves as a team progresses through the sprints. The team learns more about what requirements are needed as the endeavor progresses. This is how many endeavors evolve. The TravelEssence team knew from their previous work with Essence (during the internal demo) the value of the alpha checklists, especially in regard to helping them get their Work to the Under Control state. As shown above, for example, checklists from the Work alpha had already reminded the team of the need to make sure they had sufficiently broken the work down. This helped the team confidently estimate the work to fit within its agreed sprint.

During the sprint planning session Joel said, “I don’t think we have enough information to estimate the work involved in Item #2. Angela, can you explain more about what you mean by the toggle for recommendations based on user group?” After further discussion, the team felt they understood that all Angela wanted was a way to turn the recommendation on or off for particular user groups. This conditional check could be easily achieved just by adding several lines of code. Thus, they now understood the backlog item well enough to make an estimate (see Table 14.3).

What we have just described demonstrates how, by applying the Sprint Planning activity that is part of the Scrum practice, the team members were better able to understand the work products.

14.8.2 Daily Scrum

The Daily Scrum is a simple activity that Scrum teams conduct every day. There are only a few guiding principles required (Figure 14.14), such as keeping the meeting to 15 minutes, having only the developers speak, and keeping the focus on answering the three main questions (what did I do since the last daily scrum, what do I plan to do next, and what obstacles am I facing).

When conducting the daily scrum, Smith’s team met at the same time and same place each day. Keeping the meeting to just 15 minutes forced Smith’s team members to be as brief as possible, and focused the team just on answering the three questions. As each team member answered the questions, the others listened, and if they heard something with which they could help, they agreed to talk further after the meeting with just the smaller group that needed to be involved in the discussion. In this way, team communication was enhanced while minimizing the time lost by all team members in attending the meeting. When someone identified an obstacle that was keeping them from getting their work done, Smith, the Scrum Master, accepted an action to work the issue, but sometimes other team members stepped up and helped when they knew how to solve the problem. Still, they didn’t

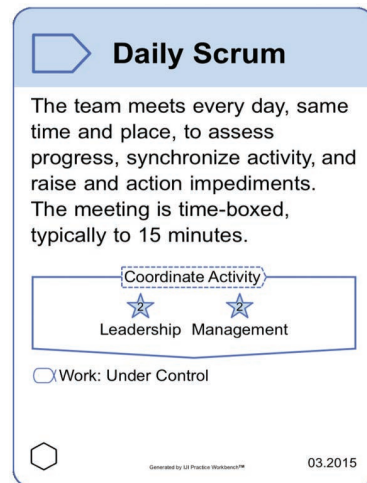


Figure 14.14 The Daily Scrum activity card.

discuss the solution in the daily scrum because they didn't want to take up the time of the other team members who didn't need to be involved in solving the problem. The daily scrum helped the team keep the Work under control. (In Figure 14.14, at the bottom of the card, the result of the activity is that the Work alpha will be in the Under Control state. When no state is specified as input to the activity it means that the daily scrum is done in the same state.)

During one daily scrum, Tom said, "I have been working since our last daily scrum on Item #4, the introductory user screen, and I am having some trouble getting it to work." Joel replied, "Tom, I have worked on introductory user screens before, so I will give you some help right after the daily scrum." From an Essence perspective, we can see from Joel's reply how their daily scrum is helping to progress the *Team* alpha's Collaborating state checklist items: "The team is working as one cohesive unit," and "Communication within the team is open and honest."

The value in documenting these simple guidelines for a daily scrum as checklist items in an activity is that they serve as reminders to the team that can help them conduct the daily scrum consistently. These checklist items can likewise be used during training and coaching sessions. They are also useful for bringing new hires on board.

14.8.3 Sprint Review

The Sprint Review is a review of the product by the stakeholders (see Figure 14.15). The focus of this review should be on demonstrating what the team produced based

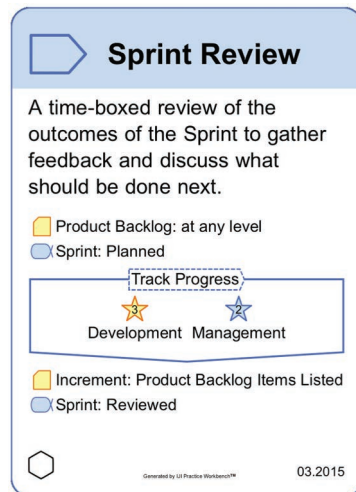


Figure 14.15 The Sprint Review activity card.

on what they committed to produce at the previous sprint planning session. At TravelEssence Angela, as the product owner, led this meeting, but she was supported by the other team members who worked on the changes related to the current sprint. To keep the meeting focused, Angela started each sprint review by going over the sprint backlog items that the team agreed to work on during the previous sprint. Then each item was demonstrated, and Angela asked the stakeholders in attendance if they agreed that each sprint backlog item that was committed to was achieved. Only sprint backlog items that were completed during the sprint were demonstrated. If something was partially completed, its demonstration was put off until the next sprint, when it could be fully demonstrated. Scrum teams do not take “partial credit” for completing part of a backlog item. If committed sprint backlog items are not completed, the product owner explains this during the sprint review and explains the plan to address the missing item. The sprint review is also an opportunity for the team members to get valuable feedback from the stakeholders. This occurs primarily at the end of the sprint review, when the product owner asks the stakeholders if they feel that the goal of the sprint was achieved. However, often stakeholders provide feedback throughout the review. In this case, at the end of the sprint review the PO summarized the result of the review and the actions the team was taking out of the review to address in a future sprint review. Input to the review is the product backlog that is used to ensure the review focuses on the items the team committed to complete. The card also shows that sprint alpha

is in the Planned state prior to the Sprint Review activity, and is progressed to the Reviewed state as an output of a successful review. Moreover, it shows that the increment work product is updated with new product backlog Items as an output of the review.

Often, when Scrum teams operate with only tacit practices, the sprint review can lose its focus, with stakeholders bringing up issues that were never planned as part of the sprint, or team members discussing the method they are following rather than the product they have produced. The value in adding a simple Sprint Review activity, or at least adding checklists, is that these checklist reminders can help the team to recall their agreed-to activities related to the sprint review. It can also help to bring new people on board, similar to what we just discussed with the daily scrum.

One Friday at TravelEssence, Angela, as the PO, started out the sprint review by explaining to the stakeholders who had come—Cheryl and Dave, since this sprint was focused on an internal release—the product backlog items that the team had committed to for this sprint, and what they were going to see demonstrated. Angela then asked Grace, Tom, and Joel to demonstrate what they had accomplished during the sprint. At the conclusion, Angela explained how they had not been able to fully implement the user screen because they ran into a few issues, but they felt they did meet their commitments for achieving the toggle for recommendations. She then asked the stakeholders in attendance for their feedback, and whether they thought the goal of the sprint had been achieved.

It is worth noting that often explicit practices and their associated activities can help teams progress multiple Essence alphas at the same time. For example, the sprint review activity just discussed, as part of the Scrum Lite practice, helped the team progress the Essence Stakeholder alpha's Involved state, checklist item "The stakeholder representatives provide feedback and take part in decision making in a timely manner." This was seen with the involvement and feedback provided at the sprint review by the stakeholders Cheryl and Dave. As another example, we can also observe from the sprint review activity how the Essence Work alpha's Under Control state, checklist items "Estimates are revised to reflect the team performance" and "Measures are available to show progress and velocity" were achieved by the team's discussion with the stakeholders on the issues they encountered and the tasks they successfully completed.

14.8.4 Sprint Retrospective

The purpose of a sprint retrospective is for the team to review how they are doing on their endeavor from the perspective of their agreed-to method, and to agree to

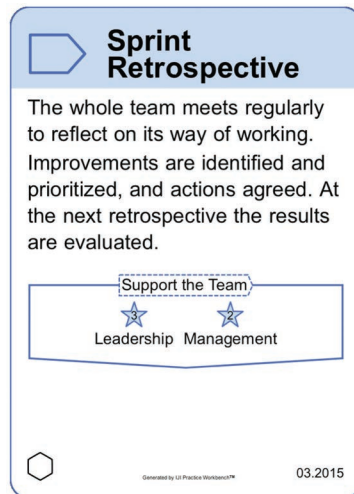


Figure 14.16 The Sprint Retrospective activity card.

improvements to their method to implement in the next sprint. The results of these improvements can be tacit or explicit, which means they may or may not require changes to practice descriptions.

There are many techniques available for conducting sprint retrospectives. There are even books that have been written just focused on this subject alone [Derby and Larsen 2006]. The value of the Sprint Retrospective is to get feedback from your development team on what is working well and what isn't working well, and get agreement with the team on what they can do differently during the next sprint to improve their method (see Figure 14.16). The Sprint Retrospective can also help to guide teams with decisions related to how to move forward with the team's suggested improvements.

A sprint retrospective could be represented in the Essence language as an activity within a larger practice, such as Scrum, or as a practice itself. For example, many organizations break their retrospectives out as a separate practice and include in the practice criteria to help teams select practical improvements that can be implemented within the next sprint. An example of such criteria are referred to as the SMART criteria, which stands for Small, Measurable, Attainable, Relevant, and Testable. These attributes are intended to be used by teams to help them assess if their agreed to improvements can be implemented within the next sprint. For instance, a team member could ask the following questions in regard to a proposed improvement.

- Is this improvement *small* enough for the team to implement it the next sprint?
- Is this improvement *attainable*? If all the team members do not have the needed skills to implement the improvement, then it might not be attainable.
- Is the improvement *relevant*? There are lots of improvements that teams could decide to make that might not be relevant to helping the team achieve their goal.
- Is the improvement *testable*? We need to know how we are going to test the improvement to say whether we achieved it or not.

In our TravelEssence case, Smith started the sprint retrospective by asking all the team members to jot down on yellow sticky notes things they thought went well during the sprint and things they thought didn't go well and could be improved. Smith then collected all sticky notes and grouped the ones that seemed to be related under larger sticky notes that said "working well" and "not working well." He then highlighted to the team that everyone felt the team was working well with respect to communication. (It is a good idea to always start a sprint retrospective by sharing with the team what is working well so the team doesn't feel like they are always just focusing on negative things.) Smith then moved over to where he had grouped a number of yellow sticky notes under the heading "not working well." He pointed out that a number of team members felt that the team could do better in future sprint planning sessions.

Smith said, "Grace, one of your sticky notes says 'sprint tasks unclear'. Could you explain what you mean by that?" Grace replied, "Sure. I don't think we are breaking our sprint tasks down enough and describing enough about what needs to be done to complete each task." The team then used the Essence Work alpha's Started state, checklist item "The work is being broken down into actionable work items with clear definition of done" to stimulate their retrospective discussion related to breaking work down. Tom said, "I agree with you, Grace. I think we need to spend a little more time discussing and agreeing what 'Done' means for each of our tasks before we commit to the sprint work." The team agreed that this would be an improvement area they would work on during the next sprint. Specifically, they made it a point to break down the work into items each with clear acceptance criteria. This was one of the reasons why they moved to user stories, which will be fully described in Chapter 15.

14.9 Reflecting on the Use of Scrum with Essence

As we saw in Part II, Essence by itself can help teams who are experienced, or are working on a simple endeavor where their practices are tacit (i.e., not written down). When your endeavor is more complex or more people are involved who have never worked together, then there is increased risk of miscommunication of what activities the team members are expected to carry out and the degree of detail expected in work products produced. There is also increased risk that different team members will assess their progress and risks differently, leading to confusion and inaccurate progress reporting to stakeholders.

14.9.1 Adding Explicit Practices

Scrum essentialization helps you to focus on the essentials of Scrum in two important ways:

1. Calling out the most important parts of the practice.
2. Making explicit what these important parts are.

For example, there are many books written about Scrum, but how do you explain Scrum quickly to a team new to Scrum? You will inevitably need to choose what to talk about and what to ignore. It is worth repeating that in this chapter, we have done that, and we call what we want to talk about Scrum Lite. Next, you have to describe the result in a way that is easy to understand and not misinterpreted by the team. We do that by essentializing Scrum Lite. Recall that when we use the term “essentialized” we mean you have described your practice using the Essence kernel and language.

Thus, an essentialized Scrum Lite can provide guidance in the essentials of conducting certain activities such as Sprint Retrospective and Sprint Review. There are many different ways teams can conduct a sprint retrospective [Derby and Larsen 2006] that goes beyond what is essential. As an example, they can ask each team member to write down what went well and what didn’t go well during the last sprint. This helps the team decide where they can improve for the next sprint.

14.9.2 Visualizing the Impact of the Scrum Lite Practice

Let’s see how essentialization helps teams see more clearly where gaps exist in their own practices, using the Essence kernel as a reference. This leads them to see where specific new explicit practices may be needed or improvements made to their existing explicit practices.

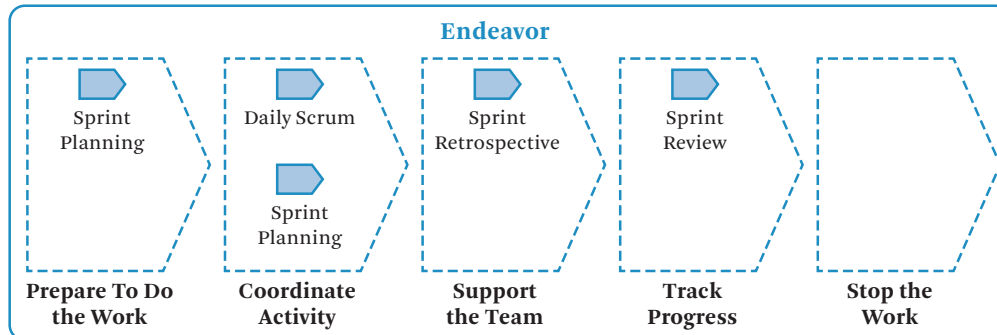


Figure 14.17 Endeavor activity spaces partially filled with Scrum Lite activities.

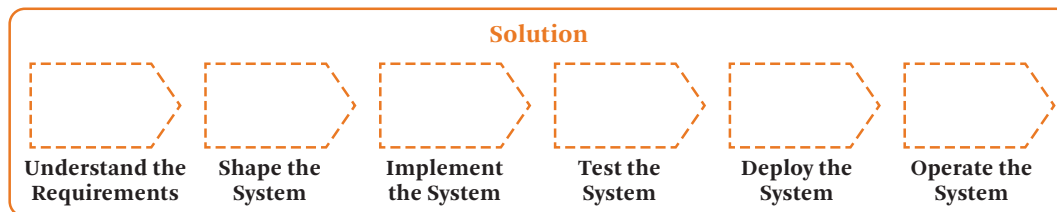


Figure 14.18 Solution activity spaces (not addressed by Scrum Lite).

To demonstrate what we mean, Figure 14.17 shows all the activity spaces in the endeavor area of concern. It also shows the activity spaces that are populated by Scrum Lite activities. Note that the Sprint Planning activity occupies two activity spaces: Prepare to Do the Work and Coordinate Activity. sprint planning has the dual purpose of creating an initial product backlog and a sprint backlog from which the team can start working, and iteratively updating the product and sprint backlogs to keep the team going.

As can be seen, Scrum Lite only occupies four out of the five activity spaces in the endeavor area of concern. In particular, Stop the Work is empty. This activity space is about concluding the work. These are outside the scope of the Scrum Lite practice and thus show the team where they have gaps.

Scrum Lite also does not provide any guidance on other areas of concerns (i.e., the customer and solution areas)—nor does Scrum. As an example, Smith and his team did have issues with how to work effectively with regard to activities in the solution area of concern (see Figure 14.18) and hence needed some explicit guidance. Because the team quickly observed gaps in their requirements practice, it led them to apply further practices: User Story Lite, and then Use Case Lite.

14.9.3 Value of Being Precise

The idea of practices is not new. It has been around for maybe 50 years. As we explained early in Part I, and will review at length here, in the past, we have seen two major problems. First, practices have been applied in an ad hoc manner. Sometimes different authors use different words to mean the same thing. Or perhaps, a single word has more than one meaning, or two words may overlap, but have some major discrepancy at the same time. This meant that practitioners had to relearn the vocabulary when moving from working with one practice to working with another. What the Essence standard attempts to do is to eradicate this kind of unnecessary confusion. The second problem is that sometimes practices and methods are too rigorous. For example, some quite successful methods were described using thousands of pages. The intention was good, but it didn't work in practice. There is a law of nature: "Most people don't read big books." Thus, when developing Essence, the following points were considered for how individuals in the team/organization should learn and improve.

1. Focus on the essentials. This is the gist of a practice, maybe 5% of what an expert knows of the practice but enough to participate in a team.
2. Use a very simple intuitive visual language to describe guidance of practices, alphas, activities, etc.
3. Provide a simple way to access additional guidelines, for example using links to books, teaching materials, and tools from the essentials.
4. Keep practices separate but make them composable with other practices to form methods.
5. Keep the practices in a library and let the teams that develop software improve them easily and quickly when they do retrospectives.

Regarding the first three points above, what has been done is to be precise and explicit, and to give the essentials a certain level of rigor using the Essence language. The goal is to facilitate more effective communications and guidance, so that team members can have more time to do the actual work.

This precision is made possible not only through the language and its usage but also by focusing on the essentials. Thus, practice descriptions following this approach are very concise and intuitive to practitioners. It serves as a reminder, and helps teams get started with conversations. The Scrum Lite practice is just 12 cards (see Figure 14.19), one card for each element in Figure 14.3.

With an understanding of the Essence kernel as presented in Part II, a practitioner only needs to have a small deck of 12 cards to begin his/her journey under-

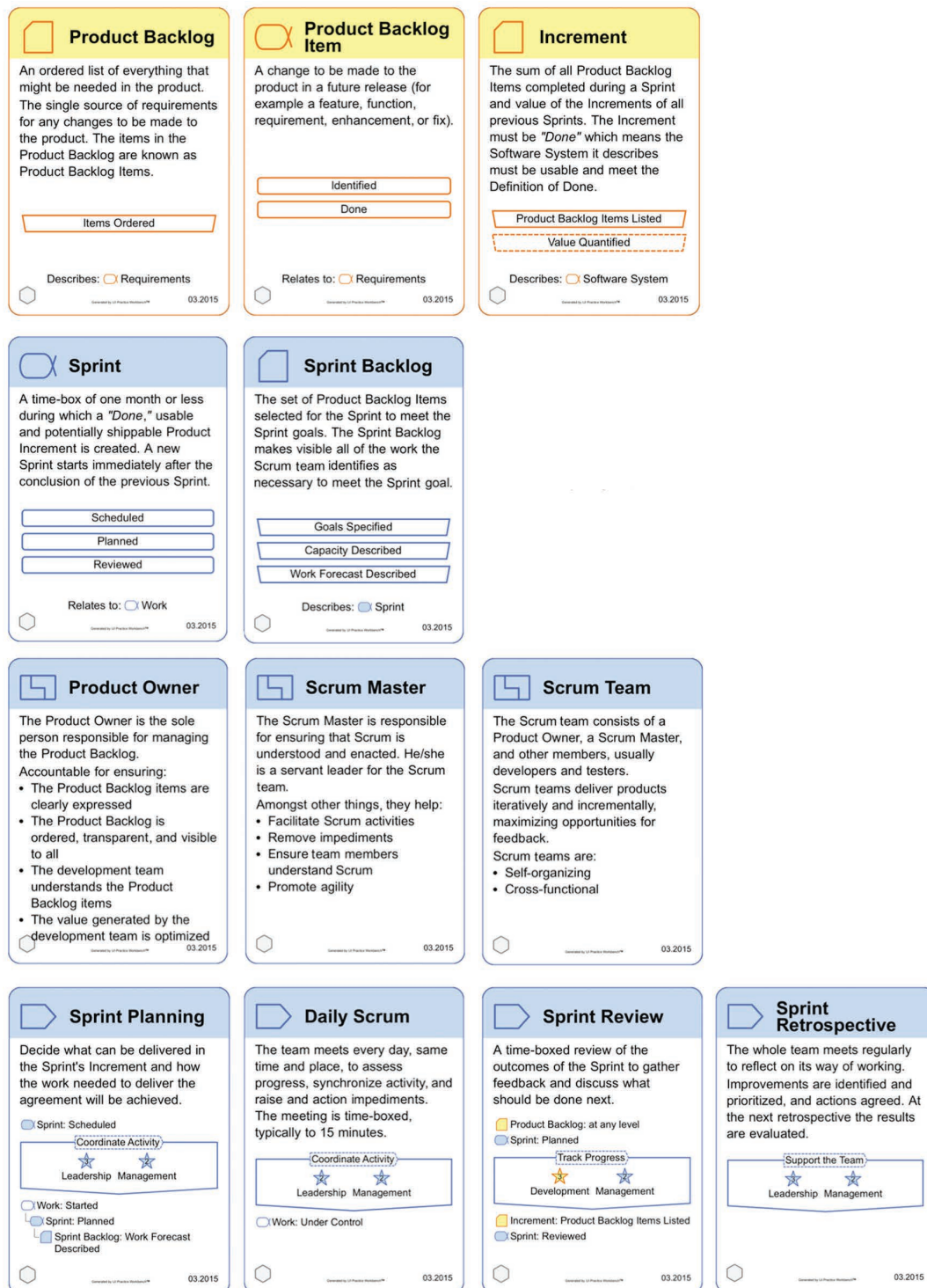


Figure 14.19 Scrum Lite as a deck of 12 cards.

standing, learning, and applying a new practice. If you want to give more guidance to the practitioners, more detailed descriptions can be made available. Associated with each card there could be a 2–4 page document with further details, guidance, hints, tips, and common mistakes. All the cards mentioned in this book are available for download on the book’s website at <http://semat.org/web/book/>.

Regarding the last two points above, Essence provides a mechanism for teams to select and compose practices. Our work with practices has yielded a sizeable set of them. We will discuss this further in Part IV, when we talk about how Essence can scale to large organizations.

So again, our recommendation to software engineering students is to learn the kernel, learn the language, symbols, and agreed terms, to lay the foundation for learning and comparing the multitude of practices in software engineering. In the remainder of this part of the book, we will demonstrate how Smith’s team—with the help of precise and explicit practices—embarked on their journey to successful software delivery and personal growth.

What Should You Now Be Able to Accomplish?

After studying this chapter, you should be able to

- explain the benefits of the Scrum practice;
- explain how TravelEssence adopted and applied Scrum and what benefits they achieved, together with the benefits implied by using the Scrum practice in an essentialized form;
- explain why organizations often mandate specific practices and tools;
- list and explain the alphas, work products, activities, and patterns of Scrum;
- explain the concept “Definition of Done” used in Scrum;
- apply Scrum Lite practice;
- name relevant questions to ask during development with Scrum (e.g., “Were the items selected for this sprint properly prepared?”); and
- explain SMART criteria for a practice (what do the letters stand for?).

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 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. 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.