DOI:10.1145/2380656.2380670

Article development led by acmqueue
queue.acm.org

A thinking framework in the form of an actionable kernel.

BY IVAR JACOBSON, PAN-WEI NG, PAUL E. MCMAHON, IAN SPENCE, AND SVANTE LIDMAN

The Essence of Software Engineering: The SEMAT Kernel

EVERYONE WHO DEVELOPS software knows it is a complex and risky business, and its participants are always on the lookout for new ideas that will lead to better software. Fortunately, software engineering is still a young and growing profession that sees innovations and improvements in best practices every year. Just look, for example, at the improvements and benefits that lean and agile thinking have brought to software-development teams.

Successful software-development teams need to strike a balance between quickly delivering working software systems, satisfying their stakeholders, addressing their risks, and improving their ways of working. For that, they need an effective thinking framework that bridges the gap between their current way of working and any new ideas they want to adopt. This article presents such a thinking framework in the form of an actionable kernel, which could benefit any team wishing to balance their risks and improve their way of working.

Work on the kernel, the essence of software engineering, was inspired by and is a direct response to the Software Engineering Methods and Theory (SEMAT) call for action (see Figure 1). It is, in its own way, one small step toward redefining software engineering.

SEMAT was founded in September 2009 by Ivar Jacobson, Bertrand Meyer, and Richard Soley, who felt the time had come to fundamentally change the way people work with software-development methods.[3,4,8] They wrote a call for action statement, which in a few lines identifies a number of critical problems, explains why there is a need to act, and suggests what needs to be done. The call for action is:

Some areas of software engineering today suffer from immature practices. Specific problems include:

- The prevalence of fads more typical of the fashion industry than an engineering discipline;
- The lack of a sound, widely accepted theoretical basis;
- The huge number of methods and method variants, with differences little understood and artificially magnified;
- The lack of credible experimental evaluation and validation; and
- The split between industry practice and academic research.

The SEMAT call for action's assertion that the software industry is prone to fads and fashions has led some people to assume that SEMAT is resistant to new ideas. This could not be further from the truth. As you will see in this article and in a soon-to-be-published book (*The Essence of Software Engineering—Applying the SEMAT Kernel*),[6] SEMAT supporters are very keen on new

ILLUSTRATION BY PETER CROWTHER ASSOCIATES

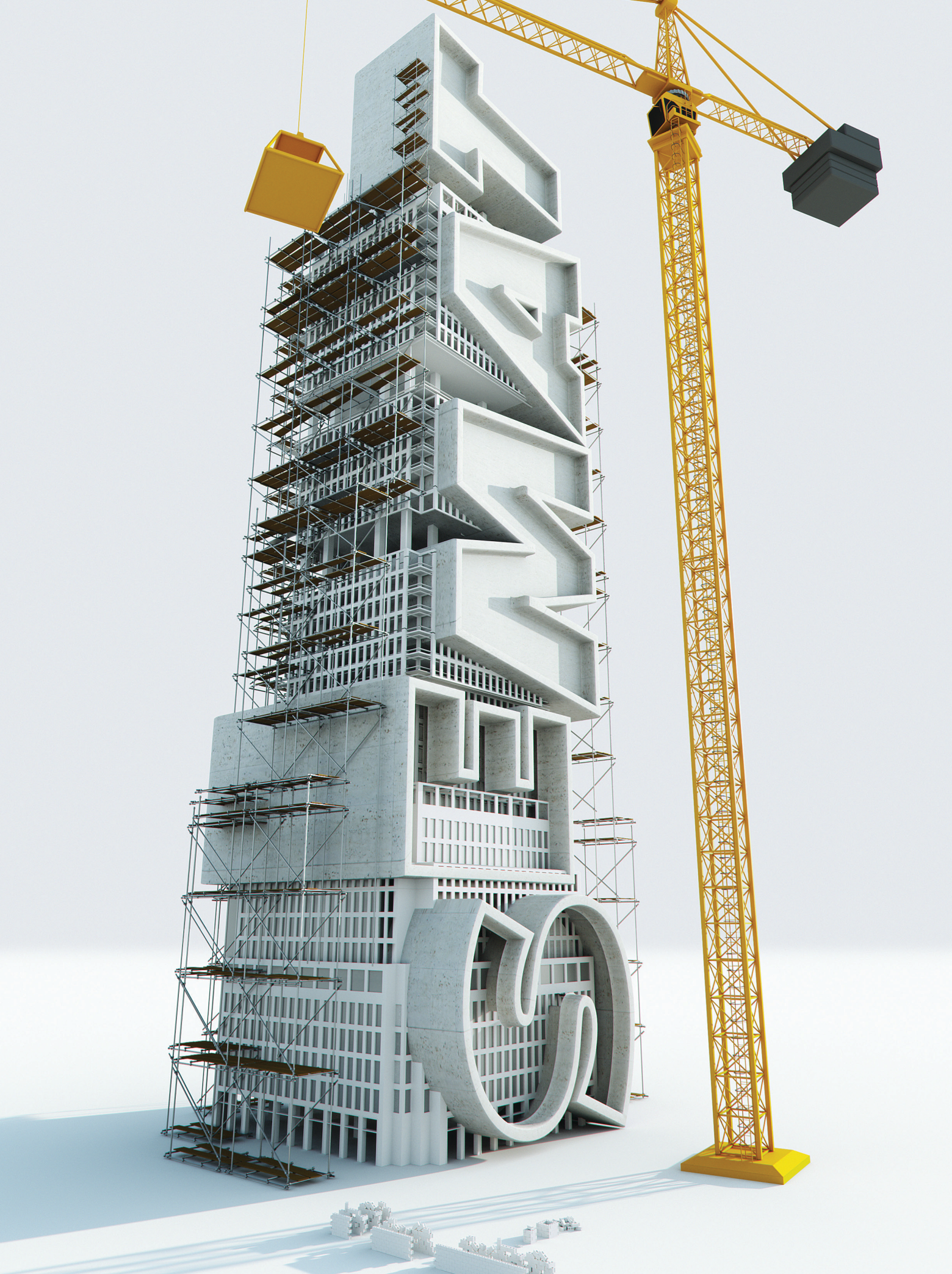


Figure 1. Things to work with.

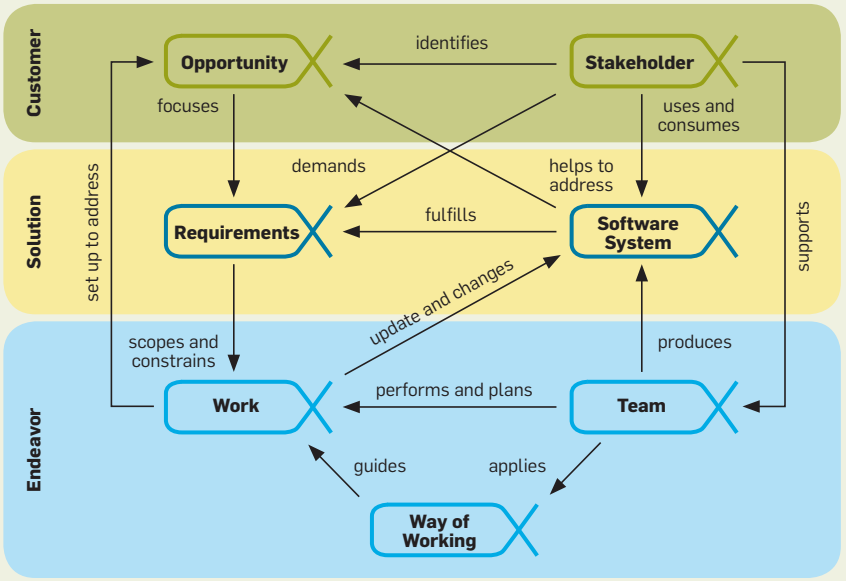
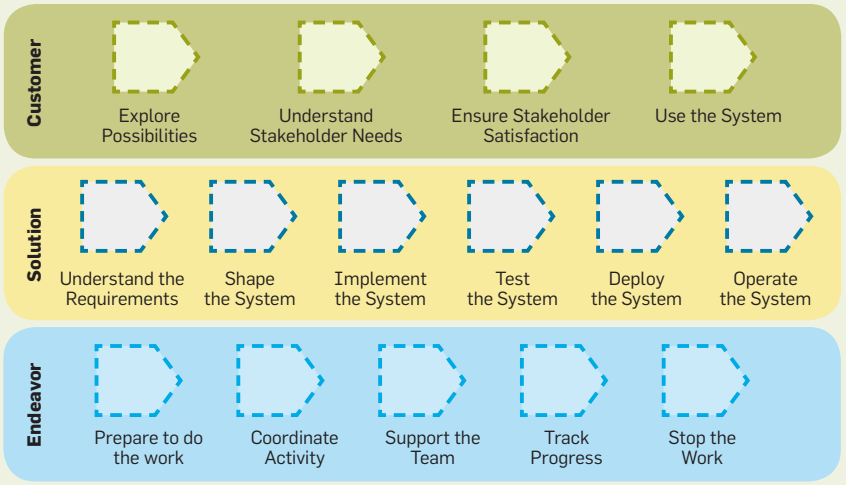


Figure 2. Things to do.



ideas. What they are against is the non-lean, non-agile behavior that comes from people adopting inappropriate solutions just because they believe these solutions are fashionable—or because of peer pressure or political correctness.

SEMAT supports a process to redefine software engineering based on a solid theory, proven principles, and best practices that:

- Include a kernel of widely agreed-upon elements, extensible for specific uses;
- Address both technology and people issues;
- Are supported by industry, academia, researchers, and users; and
- Support extension in the face of changing requirements and technology.

The SEMAT call for action received a broad base of support, including a growing list of signatories and supporters (http://www.semat.org). In February 2010 the SEMAT founders developed the call for action into a vision statement.[5] In accordance with this vision, SEMAT focused on two major goals: Finding a kernel of widely agreed-upon elements, and defining a solid theoretical basis.

To a large extent these two tasks are independent of each other. Finding the kernel and its elements is a pragmatic exercise requiring experienced software developers with knowledge of many of the existing methods. Defining the theoretical basis is academic research and may take many years to reach a successful outcome.

The Power of the Common Ground

SEMAT's first step was to identify a common ground for software engineering. This common ground is manifested as a kernel of essential elements that are universal to all software-development efforts, and a simple language for describing methods and practices. The kernel was first published in the SEMAT OMG (Object Management Group) submission.[2,9] As shown in figures 1 and 2, the kernel contains a small number of "things we always work with" and "things we always do" when developing software systems. Work is also ongoing to define the "skills we always need to have," but this will have to wait until future versions of the kernel.

More than just a conceptual model, the kernel provides:

- A thinking framework for teams to reason about the progress they are making and the health of their endeavors;
- Common ground for the discussion, improvement, comparison, and sharing of software-engineering methods and practices;
- A framework for teams to assemble and continuously improve their way of working by the composition of separately defined, and sourced, practices;
- A foundation for defining practice-independent measures to assess the quality of the software produced and the methods used to produce it; and
- Most importantly, a way of helping teams understand where they are, what they should do next, and where they need to improve.

The Big Idea

What is it that makes the kernel more than just a conceptual model of software engineering? What is it that is re-

ally new here? This can be summed up in its founding principles (see Figure 3), which really bring out three unique features of the kernel: it is actionable; it is extensible; and it is practical.

The kernel is actionable. A unique feature of the kernel is the way that it handles the "things to work with." These are captured as *alphas* rather than work products (such as documents). An alpha is an essential element of the software-engineering endeavor—one that is relevant to an assessment of its progress and health. As shown in Figure 1, SEMAT has identified seven alphas: opportunity, stakeholders, requirements, software system, work, way of working, and team.

The alphas are characterized by a simple set of states that represent their progress and health. As an example, the software system moves through the states of architecture selected: demonstrable, usable, ready, operational, and retired. Each state has a checklist that specifies the criteria needed to reach the state. These states make the kernel actionable and enable it to guide the behavior of software-development teams.

The kernel presents software development not as a linear process but as a network of collaborating elements that need to be balanced and maintained so teams can make effective and efficient progress, eliminate waste, and develop great software. The alphas in the kernel provide an overall framework for driving and progressing software-development efforts, regardless of the practices applied or the philosophy followed.

As practices are added to the kernel, alphas will be added to represent the things that either drive the progress of the kernel alphas or inhibit their progress. For example, the requirements alpha will not be addressed as a whole but will move forward item by item. The progress of the individual requirement items will either drive or inhibit the progress and health of the requirements alpha. The requirements items could be of many different types: for example, features, user stories, or use-case slices, all of which can be represented as alphas and have their states tracked. The benefit of relating these smaller items to the coarser-grained kernel elements is that it allows the tracking of the health of the endeavor as a whole. This provides a necessary balance to the lower-level tracking of the individual items, enabling teams to understand and optimize their ways of working.

The kernel is extensible. Another unique feature of the kernel is the way it can be extended to support different projects (for example, new development, legacy enhancements, in-house development, offshore development, software product lines, and so on). The kernel allows you to add practices, such as user stories, use cases, component-based development, architecture, pair-programming, daily stand-up meetings, self-organizing teams, and so on to build the methods you need. For example, different methods could be assembled for in-house and outsourced development or for the development of safety-critical embedded systems and back-office reporting systems.

Practice separation is the key idea here. While the term *practice* has been widely used in the industry for many years, the kernel has a specific approach to the handling and sharing of practices. Practices are presented as distinct, separate, modular units, which a team can choose to use or not to use. This contrasts with traditional approaches that treat software development as a soup of indistinguishable practices and lead teams to dump the good with the bad when they move from one method to another.

The kernel is practical. Perhaps the most important feature of the kernel is the way it is used in practice. Traditional approaches to software-development methods tend to focus on supporting process engineers or quality engineers. The kernel, in contrast, is a hands-on, tangible thinking framework focused on supporting software professionals as they carry out their work.

For example, the kernel can be touched and used through cards (see Figure 4).[7,10] The cards provide concise reminders and cues for team members as they go about their daily tasks. By providing practical checklists and prompts, as opposed to conceptual discussions, the kernel becomes something the team uses on a daily basis. This is a fundamental difference from traditional approaches, which tend to overemphasize method description as opposed to method use and tend to be consulted only by people new to the team.

Cards provide concise descriptions that serve as reminders for team members. They can keep the kernel as

Figure 3. Guiding principles behind the kernel.

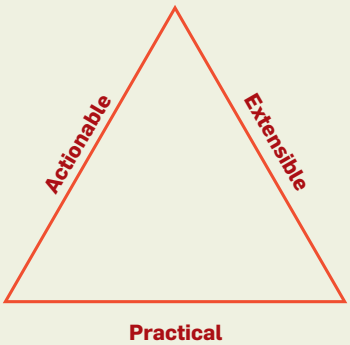
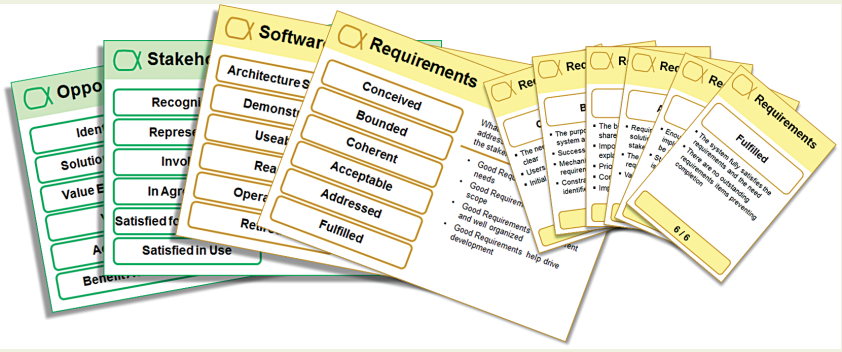


Figure 4. Cards make the kernel tangible.



a small deck of cards in their pockets, which they can easily pull out to discuss the current status of development and the work assignment and collaboration among team members. Teams can also discuss areas of improvement by referring to the cards. Thus, the kernel is not merely a heavyweight description of what a team needs to do. Rather, it forms an essential part of what they are doing each day.

The kernel in action. The kernel has many applications in software professionals' everyday lives. They include:

- Running an iteration (or sprint).
- Running the entire development from idea to product.
- Scaling to large organizations and complex software-development endeavors.

The first application, planning an iteration, is used here as an example of what a team can do with the kernel. The others are covered fully in *The Essence of Software Engineering—Applying the SEMAT Kernel*.[6]

The example presented here assumes that a company has very little in the way of formal processes. In the past it has relied on having skilled and creative individuals on experienced teams, but the company is now growing and has many new hires. These new employees, mostly fresh out of university, have good technical skills—for example, in programming languages—but are less equipped in other aspects of software development, such as working with stakeholders to gain agreement on requirements.

This company has a development team that is responsible for making a mobile social-network application that lets users share and browse ideas, photos, and comments. The team began with only two developers, Smith (the team leader) and Tom, both of whom are familiar with the kernel. They are later joined by two more developers, Dick and Harriet, who are new to the job and have no previous knowledge of the kernel. Success to team leader Smith means more than functionality, schedule, and quality. This team ran development iteratively. You can think of planning an iteration as follows:

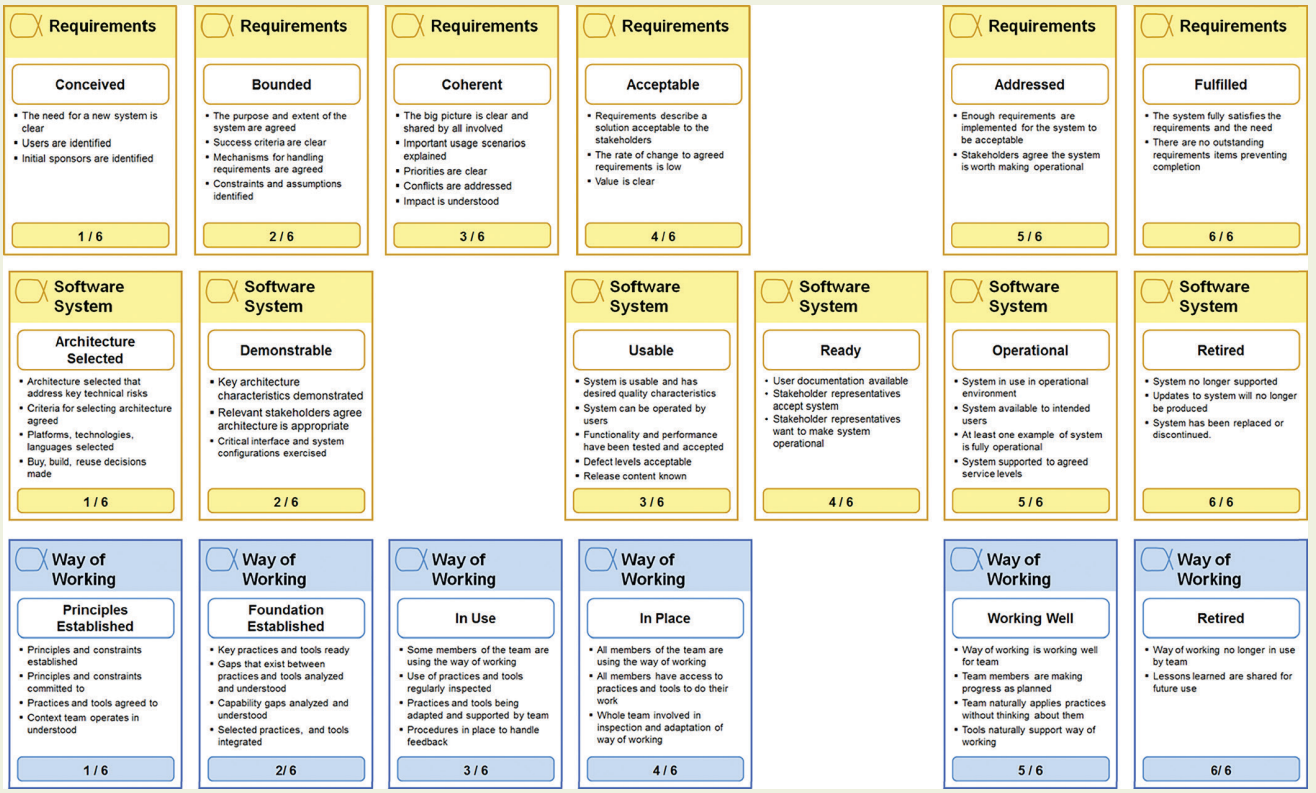
1. Determine where you are: Work out the current state of the endeavor.
2. Determine where to go: Decide what to emphasize next and what the objectives of the next iteration will be.
3. Determine how to get there: Agree on the tasks the team needs to do to achieve the objectives.

Determine Where the Team Is Using the Kernel

Let's assume that Smith and his team are six weeks into development. They have provided an early demonstration of the system to their stakeholders, who are pleased and provide valuable feedback. The system is not yet usable by end users, however. You can use the kernel to do this in a number of ways. If you are using alpha state cards, then you can do a walk-through as follows:

1. Lay out the cards for each alpha in a row on a table with the first state on the left and the final state on the right.
2. Walk through each state and ask your team if you have achieved that state.
3. If the state is achieved, move that

Figure 5. The team uses the alphas to determine the current states.



state card to the left. Continue with the next card until you get to the state that your team has not yet achieved.

4. Move this state card and the rest of the pending state cards to the right.

Figure 5 shows the states Smith's team has achieved on the left, and those not yet achieved on the right. For simplicity, Figure 5 shows only three of the kernel alphas.

Determine where to go with the kernel. Once the team agrees on the current alpha states, the members discuss what the next desired "target" states are to guide their planning. The team agrees to use the immediate next alpha states to help establish the objectives of the next iteration, as shown in Figure 6.

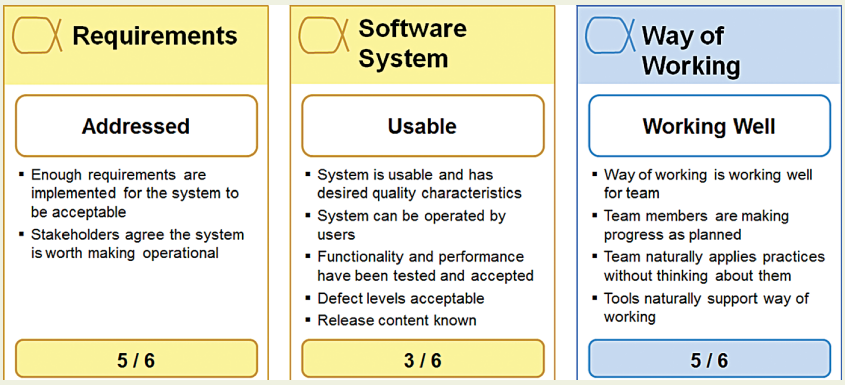
The name of the alpha state supplies a hint to understanding what needs to be achieved to reach a state. Team members can find out more by reading and understanding the alpha-state checklist. By going through the states one by one for each alpha, a team quickly becomes familiar with what is required to achieve each state. In this way the team learns about the kernel alphas at the same time as they determine their current state of development and their next target states.

Determine how to get there with the kernel. Smith and his team look at the next target states and agree that they need to establish some priorities. First, they need to determine their *way of working*: *working well*; then the *software system*: *usable*; and finally *requirements*: *addressed*. The reason is simple; not having the way of working: working well would impede their attempts to get the *software system*: *usable*. In addition, they agree on the priority for the missing requirement-items necessary to achieve the *requirements*: *addressed* state.

Smith and his team next discuss what needs to be done to achieve these states, as shown in the accompanying table. By going through the target alpha states, Smith is able to determine a set of objectives and tasks for the next iteration.

How the kernel helps in planning iterations. A good plan must be *inclusive*, meaning that it includes all essential items and covers the whole team. It must also be *concrete*, so it is actionable for the team. The team must also have

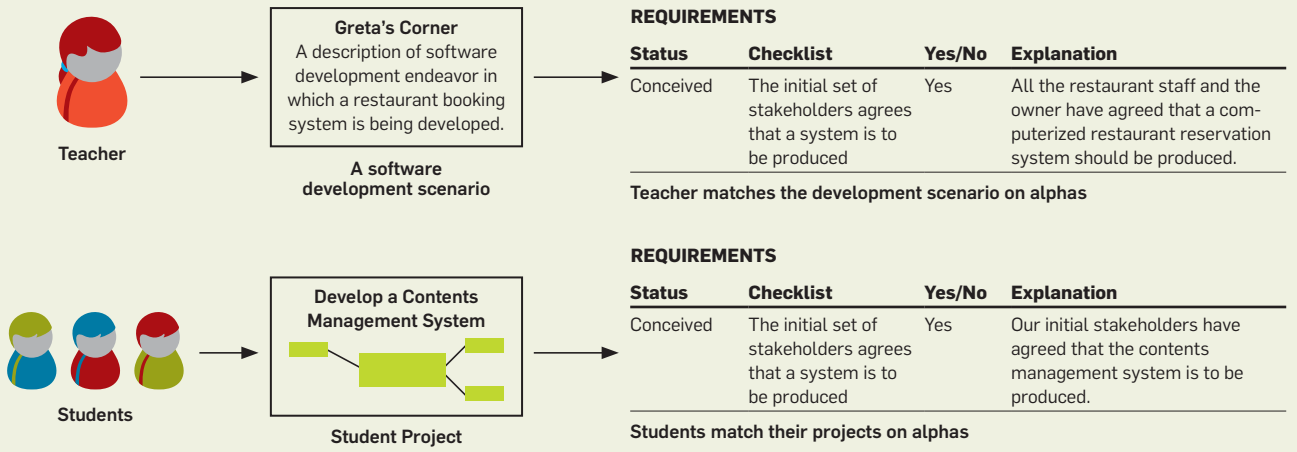
Figure 6. The selected next steps.



How the team achieves states.

Target State	How they planned to achieve them
Way of Working Working Well • Way of working is working well for team • Team members are making progress as planned • Team naturally applies practices without thinking about them • Tools naturally support way of working 5 / 6	Both Dick and Harriet agreed that they had difficulties in applying automated testing. They needed help in order to make progress. Tom agreed that he had to spend time teaching them: A task was added to the iteration backlog for Tom to conduct training on automated testing for Dick and Harriet. ► Task 1. Conduct training on automated testing.
Software System Usable • System is usable and has desired quality characteristics • System can be operated by users • Functionality and performance have been tested and accepted • Defect levels acceptable • Release content known 3 / 6	This state reminds us that the software system must be shown to be of sufficient quality and functionality to be useful to the users. So far, Smith's team had been testing within its development environment. Now, it had to conduct tests within an acceptance-test environment, which they had yet to prepare. This resulted in the following task: ► Task 2. Prepare acceptance-test environment. Smith's team had to bring all requirement-items currently demonstrable in the system to completion. This meant that each requirement-item must be fully tested within the acceptance-test environment. ► Task 3. Complete requirement-item A: "Browse online and offline." ► Task 4. Complete requirement-item B: "Post comment (online and offline)." ► Task 5. Complete requirement-item C: "Browse album."
Requirements Addressed • Enough requirements are implemented for the system to be acceptable • Stakeholders agree the system is worth making operational 5 / 6	This state reminds us of the need to work with stakeholders to ensure they are happy with the system produced. In our story Smith had to work with Angela, the customer representative, to determine which additional requirement-items needed to be implemented. This resulted in the additional task: ► Tasks 6: Talk to Angela and agree on additional requirements-items, fitting in the iteration, to make the system worth being operational.

Figure 7. Visualizing the pedagogical approach when teaching the SEMAT kernel.



a way to monitor its progress against the plan. The kernel helps you achieve this as follows:

Inclusive. The kernel alphas serve as reminders across the different dimensions of software development, helping create a plan that addresses all dimensions in a balanced way.

Concrete. The checklists for each alpha state hint at what you need to do in the iteration. The same checklists help determine your progress by making clear what you have done and comparing this with what you intended to do.

The Kernel in the Real World

Although the ideas presented here will be new to many of you, they have already been successfully applied in the real world by both industry and academia. In all cases they used the kernel and practices developed by Ivar Jacobson International.[1,10] Early adopters of the kernel idea include:

- MunichRe, the world's leading reinsurance company, where a family of "collaboration models" has been assembled to cover the whole spectrum of software and application work. Four collaboration models—exploratory, standard, maintenance, and support—have been built on the same kernel from the same set of 12 practices.
- Fujitsu Services, where the Apt Toolkit has been built on top of an early version of the software-engineering kernel, including both agile and waterfall ways of working.[1]
- A major Japanese consumer electronics company, where the software processes have been defined on top of an early version of the kernel, helping teams apply new practices and manage an offshore development vendor.
- KPN, where a kernel-based process was adopted by more than 300 projects across 13 programs as part of a move to iterative development. The kernel also provided the basis for a new results-focused QA process, which could be applied to all projects regardless of the method or practices used.
- A major U.K. government department, where a kernel-based agile toolset was introduced to enable disciplined agility and the tracking of project progress and health in a practice-independent fashion.

The kernel is already being used in first- and second-year software-engineering courses at KTH Royal Institute of Technology in Sweden. After students in the first-year courses conducted their projects, they went through the SEMAT alphas and matched them to their project results, under the direction of Anders Sjögren. The students had the opportunity to acquaint themselves with and evaluate the alphas and gain insight into the project's progress and health. In the second-year courses, run by Mira Kajko-Mattsson, the students were asked to use the SEMAT kernel when running their projects along with the development method they followed. As shown in Figure 7, Kajko-Mattsson created a software-development scenario and evaluated it for each alpha, its states, and the state checklist items. The students were then asked to do the same when conducting and evaluating their projects.

The experiences of these courses provided valuable lessons. For example, the kernel assures that all the essential aspects of software engineering are considered in a project. By matching the project results against the kernel alphas, the students could easily identify the good and bad sides of their development methods. The kernel also prepared students for future software-engineering endeavors with minimal teaching effort. By following all the kernel alphas, the students could learn the total scope of the software-engineering endeavor and thereby see what would be required of them in their future as professionals.

How the kernel relates to agile and others. The kernel can be used with all the popular management and technical practices, including Scrum, Kanban, risk-driven iterative, waterfall, use-case-driven development, acceptance-test-driven development, continuous integration, and test-driven development. It will help teams embarking on the development of new and innovative software products and those involved in enhancing and maintaining established software products. It will help all sizes of teams from one-man bands to 1,000-strong software-

engineering programs.

For example, the kernel supports the values of the Agile Manifesto. With its focus on checklists and results, and its inherent practice independence, it values individuals and interactions over processes and tools. With its focus on the needs of professional software-development teams, it values the way of working and fulfilling team responsibilities over methods.

The kernel does not in any way compete with existing methods, be they agile or anything else. On the contrary, the kernel is agnostic to a team's chosen method. Even if a team is already using a particular method, the kernel can still help. Regardless of the method used, as Robert Martin pointed out in his foreword to *The Essence of Software Engineering*, projects—even agile ones—can get out of kilter, and when they do, teams need to know more. This is where the real value of the kernel lies. It can guide a team in the actions they need to take to get back on course, to extend their method, or to address a critical gap in their way of working. It focuses on the needs of the software professional and values the "use of methods" over "the description of method definitions" (as has been normal in the past).

The kernel does not just support modern best practices; it also recognizes that a vast amount of software is already developed and needs to be maintained. It will live for decades and will have to be maintained in an efficient way. This means the way you work with this software will have to evolve alongside the software itself. New practices will need to be introduced in a way that complements the ones already in use. The kernel provides the mechanisms to migrate legacy methods from monolithic waterfall approaches to more modern agile ones and beyond, in an evolutionary way. It allows you to change your legacy methods practice-by-practice, while maintaining and improving the teams' ability to deliver.

How the kernel will help you. Use of the kernel has many benefits for experienced or aspiring software professionals, and for the teams they work in. For example, it helps you assess the progress and health of software-development endeavors, evaluate current practices, and improve your way of working. It also helps you improve communication, move more easily between teams, and adopt new ideas. It will help the industry as a whole by improving interoperability among teams, suppliers, and development organizations.

By providing a practice-independent foundation for the definition of software methods, the kernel also has the power completely to transform the way that methods are defined and practices are shared. For example, by allowing teams to mix and match practices from different sources to build and improve their way of working, the kernel addresses two of the key methodological problems facing the industry:

- Teams are no longer trapped by their methods; they can continuously improve their way of working by adding or removing practices when the situation demands.
- Methodologists no longer need to waste time describing complete methods; they can easily describe their new ideas in a concise and reusable way.

Finally, the kernel benefits academia. The kernel provides a basis for the creation of foundation courses in software engineering that can then be complemented with additional courses in specific practices—either as part of the initial educational curriculum or later during the student's professional development. Equally as important is the kernel's ability to act as a shared reference model and enabler for further research and experimentation. C

Related articles on queue.acm.org

There's No Such Thing as a Free (Software) Lunch
Jay Michaelson
http://queue.acm.org/detail.cfm?id=1005066

Purpose-Built Languages
Mike Shapiro
http://queue.acm.org/detail.cfm?id=1508217

Open Source to the Core
Jordan Hubbard
http://queue.acm.org/detail.cfm?id=1005064

References

1. Azoff, M. Apt Methods and Tools, Fujitsu. Ovum Technology Report. Reference Code 0100032-002 (Jan. 2011).
2. Fujitsu, Ivar Jacobson International AB, Model Driven Solutions. Essence—kernel and language for software engineering. Initial submission, 2012, version 1.0.
3. Jacobson, I. and Meyer, B. Methods need theory. *Dr. Dobb's Journal*,(2009).
4. Jacobson, I., Meyer, B. and Soley, R. Call for action: The SEMAT initiative. *Dr. Dobb's Journal*, (2009).
5. Jacobson, I., Meyer, B. and Soley, R. The SEMAT vision statement, (2009).
6. Jacobson, I., Pan-Wei Ng, McMahon, P., Spence, I. and Lidman S. *The Essence of Software Engineering—Applying the SEMAT Kernel.* Addison-Wesley. (Forthcoming in Jan. 2013 but available in a prepublication version on safaribooksonline.com.)
7. Jacobson, I., Pan-Wei Ng and Spence, I. Enough of process—let's do practices. *Journal of Object Technology 6*, 6 (2007), 41-67.
8. Jacobson, I. and Spence, I. Why we need a theory for software engineering. *Dr. Dobb's Journal*, (2009).
9. Object Management Group (OMG). RFP: A foundation for the Agile creation and enactment of software engineering methods, (2012).
10. Pan-Wei Ng and Magee, M. Lightweight application lifecycle management using state cards. *Agile Journal* (Oct. 2010)

Ivar Jacobson, the chairman of Ivar Jacobson International, is a father of components and component architecture, use cases, the Unified Modeling Language, and the Rational Unified Process. He has contributed to modern business modeling and aspect-oriented software development.

Pan-Wei Ng coaches large-scale systems development involving many millions of lines of code and hundreds of people per release, helping them transition to a lean and agile way of working, not forgetting to improve their code and architecture and to test through use cases and aspects. He is the coauthor, with Ivar Jacobson, of *Aspect-oriented Software Development with Use Cases.*

Paul McMahon is an independent consultant focusing on coaching project managers, team leaders, and software professionals in the practical use of lean and agile techniques in constrained environments. He has been a leader in the SEMAT initiative since its inception.

Ian Spence is CTO at Ivar Jacobson International and the team leader for the development of the SEMAT kernel. He has introduced hundreds of projects to iterative and agile practices as well as led numerous successful large-scale transformation projects working with development organizations of up to 5,000 people.

Svante Lidman has extensive experience building high-performance enterprise software-development teams. He has held positions at Hansoft AB, Ivar Jacobson International, Microsoft, Rational Software, Objectory, among others. Since mid-2010 Lidman was the leading change agent in the largest lean/agile transition ever done in Scandinavia.