24 QUESTIONS

SEMAT and Essence:

The Why's, What's and How's to See the Difference

Ivar Jacobson
Paul E. McMahon
Roland Racko

SEMAT

Over the years our collective experience has revealed many questions on the SEMAT and Essence initiative. To bring clarity of the initiative to our readers, we have answered 24 of the most common questions.

When you understand these questions and answers, you will better understand the impact and utility of Essence. The questions and answers will give you a fresh perspective on all software development methodologies, whether they are traditional or Agile such as Scrum, Kanban, SAFe, DAD – a perspective useful for the whole software development community – industry (developers and executives) and academics (teachers and researchers). In what follows Roland asks the 24 questions and Paul and Ivar provide the answers. Our goal is to help you understand how SEMAT and Essence can:

- Guide developers in achieving measurable results and to reuse their expertise in a systematic way.
- Help executives to lead programs and projects in a balanced way without applying more governance than necessary and to develop organizations, which routinely learn from that experience.
- Allow teachers to teach software engineering in a more logical and systematic way.
- Enable researchers to use Essence as a definition of the problem they want to understand and assist their efforts to develop a General Theory of Software Engineering.

We have organized the interview results into these sections:

1. The common grounds of Essence
2. Essence kernel
3. Advantages of Checklists
4. On being a standard
5. SEMAT vs SWEBOK
6. Essence kernel supporting existing practices
7. Betterment of existing methods powered by Essence
8. Theory
9. Myth and Reality

1 ON THE COMMON GROUNDS OF ESSENCE:

1. "Why is Essence not just another method?"

Ivar: There are 100,000's of methods in the world, some of which are described and some of these described are famous such as RUP, Scrum, XP, Kanban. Essence is not one of these. Essence is just what is common for all these other methods – it is a common ground. It includes things that every method has. It has some resemblance to a method, but to really become a method you need to add things on top of it.

All of these methods mentioned above have something in common. They are all about developing software. When people develop software, they always have a number of things; ingredients that are prevalent in every software endeavor. For instance, their work always results in a software system. There is always a team. They always have a way of working. The way of working doesn't need to be formally documented in a company manual. If it's a real team they will nevertheless have a way of working.

They always do things; they always come up with what the system is going to do – we can call it requirements. They always implement the requirements, for instance by writing code. They always test. There are many things they always do. If you're really careful and identify things that are universal for all software development endeavors, you can come up with a common ground, and that's what Essence is. Essence is something we always do, we always have, and we always work with when we develop software. Thus we can say that Essence includes elements universal to all software development methods. That makes Essence agnostic to any specific method.

In contrast to Essence, a method includes things that are specific and that distinguishes it from other methods. These specific things come with practices added using Essence as a foundation. For instance, there are many ways for users to write requirements. You could just write the requirements as a functional specification. You could use structured analysis. Or you could do use cases or user stories. There are many different ways of doing requirements. The specific way a method is captured in a practice, which would then be described "on top of Essence."

2. "What is the value in identifying a common ground if it always exists?"

Paul: The common ground provides a common reference point that can help ensure essentials are not missed. Essence adds states and checklists to help assess where we are and where we need to focus next with respect to the common ground. These states and checklists give a common approach to measurement, which is something we have never had before and something the software engineering community clearly needs. The states and checklists are a key discriminator of Essence from other software frameworks. They provide critically important value beyond the common ground itself.

3. "How can Essence become popular (and generally accepted) without being seen as just another fad?"

Paul: Of course, people who just look at Essence on the surface can wonder if this is just another new method of the same type we have had for the last 20 years. However, people who prudently study Essence will understand that this does not directly compete with existing methods. Instead, it codifies what we have been doing for so many years. It makes it possible to describe existing and new practices based on a common ground.

Eventually, to remove this suspicion of fad, we need proof that Essence is for real; that it really can help organizations perform better at software development. This is now our focus. However, we need more organizations to use it, and then report on their results. If organizations are struggling with Essence, we need to openly hear about those struggles so they can be discussed and resolved.

While on one hand Essence is not new from the perspective that it embodies the essentials of what has been proven to work on past successful software endeavors, on the other hand, it is very new. Why? Because in Essence we have introduced a new construct to allow us to focus on the real outcome of a project, not on documents or activities, but on results.

This new construct, called alpha, is used to represent the key elements of a practice; the elements that produce real results and which you want to measure progress and health during your project. The alphas allow us to separate out the essentials from specific practices and methods. That separation makes it possible to measure and assess strengths and weaknesses of any team's current way of working.

Another new feature is a simple and intuitive human interface way to share the knowledge and the insights of Essence. We do this by using pictorial cards and by playing serious games with these cards. We are thus suggesting some new ways that can help teams coordinate their activities and assess their progress and health.

In many organizations adoption of Essence ideas will mean a culture shift and culture shifts take time. As just one example, over the last ten years we have seen how difficult culture shifts can be as organizations have realized they need to become more agile in the way they develop and deliver software solutions. We do not believe that organizations should change dramatically the path they are currently on in order to transform themselves into more agile organizations. Rather we believe that Essence can help to power their transformation by guiding their teams especially during the difficult transition stages. This is just one way that Essence can power whatever your organization is doing today, without radically changing what is already working for you.

Ivar: Let me add that the risk that Essence will be seen as another fad reduces with every company that applies it. Several companies have successfully used it, for instance Red Hat, Fujitsu, and Munich Re to name a few such companies, but there are others such as large telecommunication vendors, mobile phone operators, electronic equipment vendors, and manufacturing companies.

Moreover, apart from being able to describe existing and new practices based on a common ground you can also take an existing method and essentialize it, meaning capture the essence of that method.

The value of 'essentialization' is that people can learn a new practice in a very easy way, compare it with other practices, compose it to a method (with many proven practices) and modify/change their method as time goes by. Applying Essence makes it fundamentally easier to govern the number of methods you have in your organization so you create an effective learning organization.

2 ON ESSENCE KERNEL:

4. "Is Essence kernel inherently 'waterfallish'?"

Ivar: Of course the logical answer is: no it's not. Because if it was, Essence would not be a common ground. Essence can be used for all kinds of methods, and of course nowadays, the most popular methods for new software development are agile methods. There are things that have to be done in some sequence, but these things are usually done within a short period of time – within an iteration or within a sprint. For instance, testing includes specifying what to test. In some agile methods you first write the test cases and let them work as requirements. And then you test to ensure that you meet these test cases. Thus, some activities are naturally done in sequence, but the sequences are within a much smaller scope than before, within an iteration.

5. "When I hear this alpha term my head starts to spin. I don't get it. Why didn't you pick a name that the software engineering community intuitively could understand?"

Paul: We intentionally chose a new name because there was no existing term that represented the meaning of what alphas are. The alpha construct is new and didn't exist before. Many people, when they first read about alphas, think they are just abstract work products, but that is not what they are.

Alphas, the core dimensions of a software process, exist whether there are tangible work products or not. The alphas are the critical aspects that we need to monitor and progress to ensure our endeavor is successful. They can be likened to the dashboard in your car in that they help you see where there may be trouble ahead so you can respond quickly to avoid that trouble. For example, if you have conflicting requirements you can't pass all the checklist items to get to the Requirements Alpha state called Coherent. You have to attend to the problem first.

6. "Looking at the Requirements alpha and its set of states some people get the impression that it is 'waterfallish' because of the bounded state. What do you tell them?"

Paul: We have tried to find a better word to describe the state that exists between Requirements Conceived and Requirements Coherent. In the Conceived state we have agreed there is a need for a new system. In the Coherent state we have worked through conflicting requirements and we understand the priority of the requirements.

However, before you get to Coherent we believe there is a state where a shared understanding of the extent of the solution is achieved. We have called that state Bounded. We understand this term is misunderstood by some people, but we have not found a better word to communicate this state. Most importantly, the Bounded state does not imply that the requirements cannot continue to evolve. Instead, it implies that there is a shared understanding of the extent of the solution.

7. "Is there a distinction between what Essence is and what a kernel is?"

Ivar: Yes. Essence is a kernel for software development endeavors. Thus, Essence is a specific kernel. There could be other kernels. For instance, there will be an extended kernel for systems including hardware and peopleware – a kernel for systems engineering. So there could be other kernels.

What we have seen is the Essence kernel developed for software endeavors can be easily modified to also work for system engineering. Although Essence is inherently more generic, we focus on software to make sure the kernel is efficient for software development.

8. "Why did you separate the Team from Stakeholders? Aren't team members stakeholders too?"

Paul: Of course team members are stakeholders. This question is a common question we get, and it is an example of trying to use the kernel as a way to physically partition your project. It is a mistake to use Essence to physically partition your project because that is not the purpose of Essence. Essence is a set of critical aspects that help you monitor and progress your endeavor. We separated the Team from Stakeholders because they are both essential to successful software development. They both need to be monitored and progressed. They each deserve their own attention because they each have separate potential issues that can arise and will require action to keep them both healthy.

9. "Risks are essential on all software endeavors. So why isn't risk an alpha?"

Paul: We agree that risks are essential on all software endeavors and we discussed the potential of including risk in the kernel. But the trouble we had was trying to decide where it could go.

Should it be in one of the existing alphas, for instance? Should it be its own alpha? After discussion, we decided to keep risk out of the kernel. By keeping it out of the kernel all the alpha states and checklists can be used to help decide where risk exists in an endeavor. For example, if you are having trouble achieving the Requirements Alpha Coherent state because there are conflicting requirements that you are having trouble solving, then this is an indication you may have a requirements risk.

3 ON ADVANTAGES OF CHECKLISTS:

10. "Why do you think states with checklists are a good way to measure progress and health?"

Ivar: Checklists are immensely powerful for many different professionals, even very well educated such as pilots and surgeons. Before taking off with a flight the pilots have to go through checklists for flight controls, engine, fuel, electrical, etc. The surgeon has to go through a surgical safety checklist specified by the World Health Organization with 19 questions such as 'has the patient any known allergies', 'has the patient confirmed his/her identity...'. Introducing the surgical safety checklist reduced the deaths by surgical errors by 47%. Thus, simple checklists save lives, why wouldn't they save software projects?

11. "Checklists have been around forever and ever and not given a value expected. Why do you think your checklists are better?"

Ivar: Historically we have used checklists to identify progress. The problem with these checklists has been that they have usually been related to the fact that you have done a particular activity or that you have written a particular work product, a document or something like that. Such checklists are very easy to cheat and so not really very useful. The fact that you have written a document doesn't mean the document is valuable.

By having states representing real outcome (e.g. goals), you know you really have achieved something when you have reached a particular state, not measured in terms of written documents or performed activities. For example, the Team Collaborating state cannot be achieved just by listing the names of your team members.

It requires that the team members are communicating in an open and honest way, and that the team members are focused on their agreed mission. That use of a quality qualifier rather than a number is actually the key difference. Our checklists are measuring or identifying that you really have achieved something of value and not that you have filled in a document template or that you have performed an activity.

Thus, Essence doesn't place value on documents at all; it doesn't care that you have done some activity. Instead, Essence cares that you have achieved something of value like executable code that satisfied a need for a stakeholder. Executable code is a real result that when demonstrated to a stakeholder, they can see what it does to help them address their need. As another example, the fact that you have written a requirement specification is not something that we would use to measure progress. Instead, we ask a different question, such as; have you gotten consensus among the stakeholders that these are the requirements of the system? This is the value in our checklists beyond what past frameworks have done.

12. "The different checks in the checklists are ambiguous. What value does such checklists give?"

Ivar: That is really a very important question. If you go through a checklist for a particular state, it's not necessarily instantly clear what is meant to achieve a state. This is actually intentional. If we tried to make it unambiguous, we would have to express ourselves in a formal way and then almost nobody would understand what is meant. On the other hand, if we had no precision at all the checklists wouldn't give us any hint of the meaning and that wouldn't make them useful.

Let go back to the Bounded state we discussed earlier and look at the example checklist item in this state that says, "the stakeholders have a shared understanding of the extent of the proposed solution."

Some people might think we need a complete and consistent set of requirements that are frozen to achieve this state but that is not what is meant by Bounded.

A "shared understanding of the extent" means the stakeholders agree where the boundaries of the proposed system lie. The team needs to agree that this checklist item is achieved by discussing it within the context of their own endeavor. This is one of the reasons we refer to Essence as a "thinking framework."

Essence strikes a balance and to some extent relies on people's experience. We know that within a team, people may have different opinions about the meaning of a particular checklist item. That results in a discussion, which is extremely valuable. Eventually the team will decide on how they interpret the checklist item and take a decision about whether the item has been achieved or not as in the example referred to previously.

As another example, is there consensus about the requirements? The ambiguous word there is obviously "consensus". Does consensus mean 80% of the people like it or my boss likes it? What is the meaning of "consensus" is partly environmentally determined. That makes it a useful ambiguity. It makes it a useful ambiguity because it brings up discussion, which inevitably must be clarifying to the process. Because of that discussion the team will eventually agree and take a decision about whether we have achieved what the checklist item implies and whether we have consensus or not.

4 ON BEING A STANDARD

13. "How are you going to guarantee keeping Essence up to date now that it has become a standard?"

Ivar: We now have a standard in place and we already have achieved the first set of updates and received approval of Essence 1.1. When working the changes for Essence 1.1 we considered many changes. For example, in the requirements alpha, there is one state "bounded" that people have the wrong impression that it means the requirements cannot change. That is incorrect. The checklist should indicate that we have an understanding of where the boundary of the system is. We considered changing the name of that state, but decided after discussion to leave it since we agreed we couldn't find a better name.

We have a number of things like that we have already identified. Some time ago there was a definition of "practice" that had been changed in the last round by people who worked on the finalization task force and that change had not been reviewed by people who have been working with Essence since the beginning. That was fixed in Essence 1.1. So there are some things that need to be taken care of and will continue to be taken care of in future Essence releases. These things can be dealt with in practical ways as comments to the specification. But they can also be done while doing the work on 1.2 and follow on releases.

We also now have many people working on potential changes. The ones we have are not based on mathematics, they are based on practical experience. There may be some new elements that should belong to the kernel or there may be elements that we can do something about like combine and so on. Essence is not a mathematical result, it's based on experience. So we may come up with some changes. However, I don't think the changes will be dramatic, because the kernel, as it is now, is very similar to what has been used in many custom engagements. So it's proven. But that doesn't mean it cannot be further improved.

What today we consider essential in most software development endeavors will, as we get more and more experience, be too little. So we may want to add something that we feel we should always have. Or it may be the other way around. We may find we can simplify. Essence is not natural law. Instead, it's proven experience. Nevertheless, we don't think we will see a lot of changes as time goes by.

Still we need to guarantee to keep Essence up to date, but we also need to make sure that the changes that we introduce are stable. This is supported by OMG's formal process in working with standards. Still OMG works quite fast. We will probably see a new updated version of Essence within one year.

It's of course important that the process isn't so rigid that it doesn't allow changes to happen. It's expected that there will be improvements over time and there is a process to do that.

"..Do you think that people will be tempted to make their own version of Essence in a company and thereby start a "fork"?"

Ivar: I think it will happen. There are different ways of changing. You can add practices on top of Essence. You can actually add a "package" on top of Essence as a change and in that way you create a new kernel.

There are basically three ways to change. One is to change Essence itself. That is something that goes through a formal process so it doesn't happen without a lot of thought.

Number two is that you add kernel elements in a layer outside the kernel without changing Essence itself. In a large company, for example, you may want to add practices that are in addition to the company's traditional software development practices, like business engineering. Or perhaps you want to make the kernel useful for systems of systems or any other kind of special-purpose engineering. You can add a layer outside which has the new specific elements.

Another way is to add practices on top of the kernel to extend elements that are not in the kernel.

There is a balance to take here. If you change the kernel in any of these ways, those changed kernels wouldn't necessarily be able to serve in the current marketplace for other people.

We expect that there will be libraries of hundreds of practices that are available to be used on top of the kernel once it becomes the standard. If you change the kernel, then the foundation for these practices is changed and so re-usability of the kernel may be lost. That is the risk a company would take if you do change the kernel or layers outside the kernel. But that may be something big companies care to do and that they are not concerned with whether they can get practices from an existing practice library. I would expect smaller companies to be more careful because they would really want to get practices from a library and not have to develop them themselves.

5 ON SEMAT VS SWEBOK

14. "Why isn't SEMAT just another SWEBOK?"

Ivar: SWEBOK stands for software engineering body of knowledge. And we have a clear difference between these two initiatives. SEMAT is looking for the kernel to provide a foundation for practices; practices defined on top of the kernel that can be combined and composed to form your particular way of working. SWEBOK has a different purpose and that is to identify practices and specify them. Both initiatives have the ambition to create a library of practices. So SEMAT and SWEBOK could work well together. SEMAT could take practices defined in SWEBOK and define them on top of the kernel. SEMAT is also about using practices in daily work. So usage is very important in SEMAT. SWEBOK has nothing similar. There is a good opportunity for the two initiatives to collaborate and do something that will become stronger for each one of them.

Both SEMAT and SWEBOK are about ways of working. However, SWEBOK has no common ground concept similar to what SEMAT has developed. SWEBOK is describing every practice from the bottom using English. Furthermore SWEBOK has no support for the actual doing and monitoring in everyday work.

6 ON ESSENCE KERNEL SUPPORTING EXISTING PRACTICES

15. "Why doesn't the Essence kernel include support for iterative development (or Scrum)?"

Ivar: It depends on what you mean by "support for." On the one hand Essence absolutely does provide support for iterative development, and for Scrum and for any other practice you may be interested in. For instance, Essence can work with Scrum by enhancing it. An example of how it can do this was seen with a group of students who used both Essence and Scrum as part of a course at Carnegie-Mellon West. The students reported that Essence helped them consider issues that might be a problem, but they didn't know might be problems. When you conduct Sprint Retrospectives without an aid like Essence, the team will only consider issues that they have absorbed from Scrum or that they are aware of from their own experience. Essence helped them consider things that could be problems, but they didn't know from their own experience. This is part of the power of Essence.

On the other hand, Essence, being universal, doesn't have practice-specific support. For example, it doesn't contain any guidance that explicitly calls for "iterative development." Instead, if you want to do iterative development you will need to take an iterative development practice from the library of practices and add it on top of the kernel. Now you would see how it maps to the kernel by using the kernel alphas, states and checklists in an iterative way. One of the reasons people think Essence doesn't support iterative development is because they think all the parts of a software system have to be in the same state at the same time. This is not true. Different parts of a software system can be in different states at the same time, and you can iterate through the same state multiple times with the same part of your software system if you have defined an iterative practice on top of the kernel. So Essence absolutely does support iterative development, but it also supports waterfall development if you choose to define a waterfall practice on top the kernel that only moves through the states of each alpha one time.

It is true that we don't include iterative ideas within the kernel because that would be method specific. However, you can add practices on top of the kernel to support Scrum or any other iterative approach.

Today we really don't know how much software is developed in an iterative or scrum like fashion. But of the total amount of software that we have and still are maintaining, I would assume that most software is developed using old methods that were not iterative.

We want to have a kernel that is agnostic to any particular method, so we cannot include iterative ideas within the kernel itself. We can add practices on top of the kernel to support scrum or other iterative approaches.

There is work going on now to compare, for instance, native Scrum as defined by the fathers of Scrum with the same Scrum, but defined on top of Essence. There are several significant advantages and values to defining Scrum using the Essence kernel.

7 ON BETTERMENT OF EXISTING METHODS POWERED BY ESSENCE

16. "Why do you say that any existing method becomes a better method if "powered by Essence?"

Ivar: If you have Essence as a platform for describing your method, you are sure that you won't miss any of the essential dimensions of software development.

What we have seen is that when people don't use Essence, they forget about essential aspects of their projects. They may forget about the stakeholders, or they forget about why they are doing this endeavor in the first place. For example, what is the business case? Or what is the opportunity we are trying to exploit? Or they may forget about enhancing their way of working. Or they may forget about keeping the team healthy and growing its capability over time.

Basically with Essence, you are consciously looking at all seven dimensions, the seven Alpha's. You won't forget any of the dimensions, and especially, you don't only focus on "getting code to run."

Even if code is what we want to get eventually, we need to have the right code. You don't get the right code if you don't carefully check what the stakeholders want. And you don't get the right code if you don't have the right team with the right competencies, and so on.

17. "I think I understand the basic elements of the kernel language (e.g. alpha, alpha state, competency, activity space), but I don't understand how you put these elements together in a simple intuitive way to describe a practice, or to describe a set of practices as a method. Can you give me a simple example?"

Paul: We are working on a practice development kit that will provide guidance in how you do it, but it is actually quite simple.

We represent all the elements of the language on cards. We have cards for alphas, alpha states, activities, competencies, patterns, practices and so on. You use the appropriate type of cards to define the parts of your practice that you need. It is important to understand that there isn't just one way to represent a practice in Essence.

As an example, one could define Scrum as a practice that includes a collection of activities with the Sprint Retrospective being one of those activities. As another approach, one could define the Retrospective as a standalone practice. If Retrospective was a separate practice, it could be used - actually reused - through composition when defining the complete Scrum practice and in that way making it easier to define the Scrum practice. The Retrospective practice could of course also be reused when defining other practices – alternatives to Scrum.

However, in either case we would select the appropriate types of cards, and place on the cards key information about retrospectives such as competencies needed, basic information about how to conduct the retrospective, who should attend a retrospective, how long it should be and so on. We would also include helpful hints such as things to watch out for during the retrospective, and completion criteria.

We also should mention that just as with the alpha states, checklists become very important also when defining a practice. There is the checklist to ensure you are prepared to conduct the practice, the checklist of things to watch out for while you are performing the practice, and the checklist to ensure you have completed the practice.

So in summary, a practice will be tangible as a set of cards. A method in its turn is then a set of practices, all described on cards. Thankfully we usually only have to work with one practice at a time so the needed number of cards are not too large for a developer at any specific point in his software endeavor. Typically the number of cards required for a single practice is less than 10. For really large practices, for instance a practice that assists the team all the way from requirements elicitation to code implementation and acceptance tested software, we have seen up to 20 cards. Of course, when you have practices that require large numbers of cards there are tools that allow you to work with electronic cards.

Paul: There are two perspectives to consider when looking at how SEMAT will help software endeavors; strategic (long range), and tactical (short range).

18. "How, exactly, will SEMAT help my endeavor?"

Looking out on a strategic long-range horizon there will be libraries of practices where you can compare, and select practices that fit with your endeavor needs without constantly reinventing the same practices. This will help endeavors by minimizing the upfront preparation time to tailor processes and get ready to execute an endeavor and it will lead to better practices and method design for teams because we will have shared best practices that now can be reused.

How software professionals can create an Essence practice and contribute it to an Essence library to share with others is the focus of one of our Essence User guide working groups. This User Guide will be made available for anyone to download and use from the SEMAT web site.

From a more tactical short-range perspective, Essence can help an endeavor today by helping teams enhance whatever they are doing today in multiple ways.

First, teams can today use the Essence framework to assess potential gaps that may exist in their current way of working regardless of the degree to which their way of working is documented today. An example of how a team can do this is provided in one of our scenarios in the Essence User Guide.

Teams can also use the Essence framework today to assess where they are and where they need to focus next to be successful. They can do this at the start of an endeavor or to help them work through a specific problem they know they are facing. Examples of how teams can do this are also provided as scenarios.

Regardless of the degree to which an organization decides to formally define their practices and methods using the Essence framework, they can start using the alpha state checklists to assess their progress and risks in a more consistent way leading to improved team communication. This can help teams learn faster, coordinate more effectively and track progress more consistently.

19. "How Are Essence's States any better than the States on a Kanban Board?"

Paul: When you look closely at the states that most team's using Kanban place on their Kanban board, they focus primarily on progressing the work. The work is just one dimension of what it takes to ensure you have a successful software endeavor. The Essence Alphas help you monitor Work, but they also provide six additional essential dimensions (Stakeholders, Opportunity, Requirements, Software System, Team, and Way of Working) beyond Work that must be monitored and progressed to ensure software endeavor success.

8 ON THEORY

20. "Where's the T in SEMAT"

Ivar: One of the triggers of SEMAT was a paper 'Methods Need Theory' by Bertrand Meyer and myself. Now SEMAT has a Theory area with quite a large group of participants who are working on a General Theory in Software Engineering. It has organized workshops for three years since 2012 and published papers from these workshops in ACM SIGSOFT Software Engineering Notes. We see very promising results from this area of work.

21. " In what way does Essence support building a general theory in software engineering?"

Ivar: A general theory in software engineering (GTSE) should, as I can see it, have a definition worth the name of what software engineering is. If you look in the literature you will find many alternative definitions of the length 1-10 lines, but these definitions really don't help as a base for a theory. From a theory perspective the Essence kernel and language work as such a useful definition. However, Essence is not just a static definition, it also includes dynamics for measuring progress and health, which should help in giving predictive properties to the theory.

22. "I am a developer, and I really don't care about formal or even semi-formal methods, but I do care about my way of working. Can you show me how Essence helps my team with our way of working even if we don't care about defining our method?"

Paul: This is a great question, and we have many examples in our User Guide that demonstrates this. In fact, two of our scenarios are good examples. Neither one depends on a team's formally defined practices or method. Both scenarios demonstrate how Essence can help teams with their way of working.

In one of our scenarios a team that is using Scrum decides to use Essence in an "assessment poker" fashion. Each of the team members has a set of the Essence Cards. They use the cards to assess where they are, and where they need to focus next. This scenario shows a team how they can use Essence as an aid to help them ensure they are doing the right things regardless of what practices they are currently using or the formality of those practices.

In another scenario the team uses Essence to help with a specific problem they are facing. Specifically, the team is having trouble with a resistant stakeholder so they start by looking at the Stakeholder Alpha to figure out where they are and this leads them to figure out that they need to get a stakeholder representative appointed. From there the team then gets the new stakeholder representative involved by interviewing him, and then they discover that he doesn't see the value of the new system. This lead them to the Opportunity Alpha and the Value Established state. What you learn from this scenario is how the Essence kernel helps the team figure out the root cause of the problem and the appropriate actions that need to be taken to solve the problem.

You can get more information from the User Guide when it becomes available later this year, or from a recent Google tech talk given by Ivar Jacobson and Ian Spence at http://www.ivarjacobson.com/google_presentation/.

9 ON MYTH AND REALITY

23. "There is a common myth among business owners and practicing project managers that agile methodology will reduce software development and delivery cycle time drastically. Is it so? In the name of agile, are we giving birth to fragile software that causes more re-work and eventual holes in the entire quality aspect?"

Ivar: First, nowadays what is classified as agile is everything that is good about software development. The term agile has really lost its value. How could you not be agile? However, not all so called agile practices will give the values we hope for. Some we even should stay away from. Nevertheless, some are really good. With agile as with everything else: you need to select the practices that fit your project and that give you what you are looking for.

24. "Object orientation and component based development stood on strong philosophy of separation of concerns and compartmentalization of responsibilities. But, immature learning as well as lack of fundamentals often leads to poor conceptualization of code reuse. If code is not usable in its first instance re-usability is a far cry. Do you agree? "

Ivar: Software reuse is a complex area and much can be said about it. Together with Martin Griss, I wrote a book (Software Reuse...), which discusses the questions you raise. Briefly, I would like to say that software reuse is something you need to architect to get. It is not something you get for free. It requires architecting competencies that have not been promoted since the world became agile. However, architecture practices are on their way back and software reuse will once again be a concern for all IT executives.

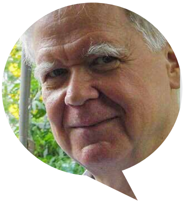
ABOUT THE AUTHORS

Dr. Ivar Jacobson is a father of components and component architecture, use cases, aspect-oriented software development, modern business engineering, the Unified Modeling Language, and the Rational Unified Process. His latest contribution to the software industry is a formal practice concept that promotes practices as the 'first-class citizens' of software development and views method (or process) simply as a composition of practices.

Dr. Jacobson is the Founder and Chairman of Ivar Jacobson International. He is also one of the three founders of the SEMAT community, the mission of which is to refound software engineering. He is the principal author of seven influential and best-selling books and a large number of papers, he was awarded the Gustaf Dalén medal ("the little Nobel Prize"), and he is an honorary doctor at San Martin de Porres University, Peru.

Paul E. McMahon (pemcmahon@acm.org), is Principal of PEM Systems (**www.pemsystems.com**) where he focuses as an independent consultant on providing coaching to project managers, team leaders and software professionals in the practical use of lean and agile techniques in constrained environments (e.g. physically distributed teams, CMMI compliance requirements, Corporate governance requirements). Before starting his independent work, Paul spent 23 years working in the software industry as a software developer, team leader and manager.

Paul has a Masters Degree in Mathematics and is a Certified Lean Six Sigma Black Belt and certified Scrum Master. He has taught Software Engineering at Binghamton University, State University of New York. He has published multiple articles and books on software development including "15 Fundamentals for Higher Performance in Software Development", and he is a co-author of "The Essence of Software Engineering: Applying the SEMAT Kernel".



Roland Racko is the president of eWyzard Inc. (**www.ewyzard.com**) which helps companies grow their businesses by using the latest in IT online marketing techniques. He has recently completed a book called "Timing Is Almost Everything." This book shows Senior Executives how to implement some of the many currently important IT technologies, like Essence, in a more effective way by carefully timing their management directives. The book also shows executives particularly effective corporate culture change tactics to smoothly introduce and adopt these technologies, including Essence, into their company.

The book, to be available in early 2016, may be ordered at **www.timingisalmosteverything.com**. Roland has over 40 years experience as an international seminar leader, magazine columnist and IT

IVAR JACOBSON
INTERNATIONAL

About Ivar Jacobson International

IJI is a global services company providing high quality consulting, coaching and training solutions for customers seeking the benefits of enterprise - scale agile software development.

We are passionate about improving the performance of software development teams, and maximizing the delivery of business value through technology.

Whether you are looking to transform a single project or program or your entire organization with lean and agile practices, we have solutions to suit your needs.

www.ivarjacobson.com

Sweden
+46 8 515 10 174
info-se@ivarjacobson.com

United Kingdom
+44 (0)207 953 9784
info-uk@ivarjacobson.com

Asia
+8610 82486030
info-asia@ivarjacobson.com

Americas
+1 703 434 3344
info-usa@ivarjacobson.com