JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2007**

Vol. 6, No. 6, July-August 2007

Enough of Processes - Lets do Practices

Ivar Jacobson, **Pan Wei Ng** and **Ian Spence** Ivar Jacobson Consulting

Abstract

All modern software development processes try to help project teams conduct their work. While there are some important differences between them, the commonalities are far greater - and understandably, since the end goal of them all is to produce working software quickly and effectively. Thus, it doesn't matter which process you adopt as long as it is adaptable, extensible, and capable of absorbing good ideas, even if they arise from other processes.

To achieve this kind of flexibility things need to change. The focus needs to shift from the definition of complete processes to the capture of reusable practices. Teams should be able to mix-and-match practices and ideas from many different sources to create effective ways of working, ones that suit them and address their risks.

In this new approach to process, which we call "EssWork", teams select practices, which are then composed and integrated seamlessly into their development environments to provide dynamic and contextualized guidance, active facilitation, and even help remove mundane tasks through clever automation.

In this paper we examine the issues facing the current generation of processes and show why we have all had enough of them. We then introduce the concept of practices and demonstrate why this is a superior approach to traditional process documentation, and the innovations needed to bring the practices to life. Finally we present EssWork and show how it can help teams realize their investment in learning, developing, and documenting best practices.

1 THE CURRENT AGE – PROCESSES

So what exactly do we mean by "process"? Process is one of those terms that are often used in our industry without any clear or consistent meaning. In this paper, we use the term "process" as an umbrella to denote a published description of a way of working. It is not a universal term because the agile camp does not use it as such. Nevertheless, in the context of this paper, we can use the term "process" understandably.

Every project team needs to have some way of working together - a way that is effective and delivers quality results on time. While every project team works differently, the way in which they work is usually inspired by what experts – internal or external -

Cite this article as follows: Ivar Jacobson, Pan Wei Ng, Ian Spence: "Enough Process - Let's Do Practices", in *Journal of Object Technology*, vol. 6, no. 6, July-August 2007, pp. 41-66
http://www.jot.fm/issues/issue_2007_07/column5

advocate. These experts have their own ways of working – their own methods, their own strategies, their own approaches, their own processes. The experts then feel the need to publish their preferred way of working in as unique and individual a way as possible, packaging their knowledge up as product – typically as a branded process, a stand-alone reference or even a process framework. There are several reasons why this is a problem.

The Problem of Denied Commonality

Each process has a few interesting gems but they are embedded in a larger package of commonalities. This follows from the facts that no individual is good at everything, and that an individual is usually only an expert on a few things

There are many different processes promoted within the software development industry. But are they really that different? If you look closely at their content, you find that there are more similarities than differences.

The reasons for this are straightforward:

- There are a number of common practices that are considered useful for all software development teams (For example, iterative and incremental development, and continuous process improvement).
- If you put two passionate experts together, their conversation will gradually progress to a debate on the subtle differences in their approaches. But the subtle differences themselves do not provide enough breadth to become full-fledged branded processes, and so they borrow silently from each other whilst continuing to debate the subtle differences.
- Everyone is looking for a silver bullet. People aren't really interested in the things that have proven their worth over time. They are just interested in what's new and fashionable, even if it is the same old things rebranded with trendy jargon and acronyms.

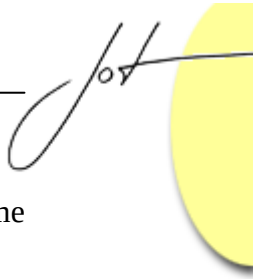
The bottom line is that the various processes have more commonality than differences. However, this can be difficult to detect as the champions of each process dress the common stuff in new words and make it sound as if everything is new.

Because everything in every new process looks new, it can be hard to really compare processes. It is even harder to mix and match - extract the gems and combine them with the gems from other processes.

The Problem of Completeness

Each process definition - large or small – wants to describe a complete process. Where the authors set out to describe a single area of the process, they inevitably end up trying to describe them all, typically in a tightly coupled and homogenous fashion.

When processes are published, their creators often find it hard to resist describing every aspect of software development. Even when they do, the methodologists who



follow in their footsteps generally try to fill in the gaps. The end result is that, as time passes, every process strives to be complete. But is this a problem?

For methodologists, it doesn't seem to be a problem. They happily "borrow" ideas from one another, adding spin, and in some cases, getting it wrong. If you speak to the people using the processes - those who do the real software development – this is a significant problem. By striving for completeness, the processes end up as brittle, all-or-nothing propositions. This makes it hard for teams to adopt just the good parts of the process and identify truly new ideas and techniques.

The desire to provide a complete process also makes the process heavier as more and more information is added to cover all of the disciplines involved. As the process evolves no one ever takes anything out because someone somewhere might need it some day (a day that never seems to come). If a process is successful, then the desire to keep it up to date and extend its market leads it to become even heavier as more information is added to expand its scope, add new techniques, and address new challenges. This leads to the need for organizations, practitioners, and methodologists to start trimming to get a lighter, more focused process. We don't think this is the way to go. You shouldn't need to spend time de-selecting the things you don't want. You should be able to start from a small, useful kernel of information, then add the guidance that you need.

The Problem of Adopting a Complete Process

Each software team has its own way-of-working (explicit or tacit), changing everything is silly, changing one thing may be smart.

Adopting a new process presents teams with many challenges, especially as they will already have their own way of working. This may be expressed explicitly as a documented process; or it may be more tacit. Within the team's way of working, there are always good practices that they will want to continue using. Other areas of the process will be weaker and lead to the desire for change. The problem with the branded processes is that they do not address this reality and require the team to change everything just to get the few new things that they want.

The same is true for organizations where, in addition to the many standard processes, they have their own in-house processes that have evolved whilst developing their software. Generally, they would like to improve these processes, but adopting another (branded) process would be too big a change. You should be able to add and drop practices as needed.

The Problem of Out-of-Sync Processes

What a team actually does never matches the adopted process as teams improvise and invent more effective ways of doing things, have to tackle problems and issues that nobody thought of when the process was selected at the beginning of the project, and never keep their process descriptions up to date.

Often, there are complaints when a team seems not to be following the defined process. However, these complaints miss the point. The point of software development is to develop good software, not to slavishly follow a predefined process.

If you have ever been involved in a process assessment, you know that what the team does rarely matches the official process. This is true of all processes. People use a mixture of ideas and practices from various sources, and although they might claim to be advocates of specific processes, few of them are actually following them to the letter.

This gap between the process as described and the process as applied is what we call the "project-process" gap. This gap causes problems for projects, teams and organizations:

- When a team is successful, it becomes difficult to spread that success to other teams. If another team takes the successful team's process and applies it, the new team won't get the same results because the process description doesn't describe what the successful team has really been doing.
- It makes it difficult to plan, estimate, monitor and control projects. The project-process gap often causes a noticeable project–plan gap to appear. It also becomes far more difficult to improve the process, compare projects or automatically capture measurements.
- Process and quality assurance becomes less effective and more expensive. The assurors spend large amounts of time looking at the process rather than at what the people actually do. The gap can even lead teams to waste a lot of time trying to achieve the illusion of process conformance - especially when they realize that they are about to be assessed.
- Teams often find that they have to spend large amounts of time filling the gap by writing large amounts of additional process documentation; for example creating local requirements management plans and configuration management plans. This local process documentation is presented separately from the process, and often contradicts or overrides large amounts of it. The end result is that no-one really knows what process the team is applying.
- It becomes far more difficult to improve or retool the project environment as it is unclear how the team is actually working. Tools are often imposed on the team that support the process description but not what the team does or needs to do .This results in either the tool never being used or a lot of time being spent learning how to configure and change the tool to do something it wasn't designed for. In some organizations, using the tool starts to become more important than doing the job; this is when you know that things have really gone wrong.

The process should be a description of what the team actually does, rather than a fictional description of what people think the team ought to be doing.

The Problem of Acquiring Knowledge

It's a law of Nature: People don't read process manuals or language specifications, they want to apply processes not read about them.

Processes are not just textual guidelines for reading pleasure. They need to be helpful and available when people are actually developing software.

Currently the majority of existing processes are published as books and static web-sites - shared resources that practitioners are expected to sit down and read. And we know that practitioners do not "read" per se - they will look the odd thing up on the Internet but not read long, scrolling pages of text. When they are actively engaged in developing software they want succinct, focused, unambiguous guidance that will immediately help them undertake their work, and not long explanations, anecdotes and academic treatises that justify and introduce the techniques they are trying to apply.

A consequence of this is that it doesn't make sense to write a lot of long and lengthy process descriptions. Process documentation needs to be presented in a new way, one that provides the practitioners with the information they need when they need it; leaving the books and knowledge bases to publish the academic foundation that the processes draw upon. Let the processes facilitate the active development of the software and the application of the practices the team selects.

The Problem of Stupid Processes

The process you have to follow doesn't actively help you do your job. You have to know what you're looking for if you want to find anything useful, and when you do find something useful you have to apply it yourself . Generally processes remind of you of all the boring things you have to do without making them any easier or less boring.

Have you ever tried to find anything useful in a process whilst you are actually developing some software? Typically the process just sits there as a passive knowledge-base waiting patiently for you to read it. It certainly doesn't interact with you and offer you appropriate and timely advice. It expects you to know exactly what you are doing and exactly what useful information it contains and exactly where it has hidden it.

When you eventually find something in the process it is typically not that useful. What you really want is some active help in the task you are trying to do and all you get is more words to read.

Processes often provide detailed instructions, for example:

- Scripts to be followed - in many cases very detailed step-by-step instructions, often going to level of what button to press or menu option to select
- Standard transformations to be applied.
- Lists of validations and qualities to be checked
- Templates and formats to apply

Wouldn't it be nice if rather than just describing these things, the process could actually play an active role in applying them? Rather than describing how you should do these things, it could actually do some of them for you: finding things, checking things, and performing mundane and trivial tasks.

Without the ability to take an active role in helping people to develop software, the process will always be nothing more than a fancy text-book, and teams will continue to struggle to realize its benefits. Process adoption will continue to rely on the presence of full-time experts, mentors and coaches who will spend a lot of their time helping with trivial work and answering the same questions over and over again - a particularly expensive way of bringing the process to life, and one, which unfortunately, doesn't scale.

What Do These Problems Mean?

Taken all together these problems are making it harder and harder for companies and teams to establish an effective way of working. The processes are becoming a barrier to change rather than an enabler. The solutions being produced don't really address the needs of development teams. A lot of good ideas are arising in the industry; other good ideas are being discarded in a vain attempt to be fashionable and to be seen as keeping up to date.

Each of the problems outlined leads to paradoxes and contradictory behavior. For example, given that people don't read process, why do so many people write them? Agile methods have been designed to rely on tacit knowledge instead of explicit knowledge; still, agile experts spend much of their time writing books and processes.

Lots of people are making lots of noise. Every practice that appears is labeled as a "best" practice. Every tool is presented as a solution to all the problems. But are we really moving forward? The industry is not really getting any better at developing software, the problems are just getting moved around. There is no firm foundation of first principles and cracks are starting to appear. The longer you spend in the industry, the more and more the success of each new process looks like another case of the Emperor's New Clothes – another triumph of spin and marketing over innovation and substance. A lot of excitement and noise is generated but if you look closely there is very little new that is actually there.

There needs to be a better way to collect and share knowledge. If it only sits in the head of a few special people it won't help us scale up. We need to break out of the patterns of the past and find new ways of capturing and sharing knowledge.

Imagine we can:

- Discover, collect and describe the knowledge of thousands of experienced practitioners
- Dramatically reduce the work to access this knowledge
- Deliver the knowledge you need, and only that knowledge, when you need it and not before

- Make practices smart, so that they actively help you to do your job better
- Empower teams to create innovative and new ways of working
- Exploit and build on existing practices rather than recycle and replace them

The problems of the past then go away and we can really start to improve the way that we develop software.

2 THE NEW ERA – PRACTICES

Phew, no wonder most developers don't like processes.

The good news, however, is that there is an alternative— practices, and there are hundreds of them. Some are generally accepted, others are unique to a particular methodology. But they all have something to offer, even though they can be hard to mix-and-match or use together. In our new approach, which we call "EssWork", these practices can be defined separately, and then composed into a simple ways of working where they are applied together. This allows teams to select the practices that they want, which are then assembled to describe their individual way of working.

What is a practice?

A "practice" provides a way to systematically and verifiably address a particular aspect of a problem.

It is important to note that:

- A practice does not attempt to address the entire problem. Rather, a practice attacks a particular aspect of the problem.
- A practice is systematic in that someone can articulate it - it is not a black art. A practice has a clear beginning and end, and tells a complete story in usable chunks.
- A practice includes its own verification, providing it with a clear goal and a way of measuring its success in achieving that goal. Without verification, the practice is not complete.

Because of these qualities, practices can be developed, learned and adopted separately, and that can be used in conjunction with other practices to create easily understood and coherent ways-of-working.

In short, a practice is 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.

Where Do Practices Come From?

The idea of practices is not a new one; practices have always existed in software development and, it seems, that every software development company in the world

promotes its own set of "best" practices. However, no one has taken the time to define exactly what practices are or how to describe them in a sensible, re-usable way. The reality is that there are hundreds of practices available for us to choose from, if only we could separate them from one another, understand the benefits they offer, and know when they should be used.

In the software development community, practices - like processes - usually come from one of three prominent camps:

- Software engineering, which includes the Unified Process [Jacobson99], Booch [Booch94], OMT [Rumbaugh91], Responsibility-Driven Design [Wirfs-Brock02], Shlaer/Mellor [Shlaer88], User-Centered Design [Constantine99], Open [Yourdon97], and Feature-Driven Development [Palmer02], among others.
- Process assurance, which includes CMMI [Chrissis06], SPICE [Emam97], Six Sigma [Barney03], and TSP/PSP [Hunphrey97]
- Agile, which includes XP [Beck04], SCRUM [Schwaber04], Crystal Clear [Cockburn04], Adaptive Software Development [Highsmith00] and Organizational Patterns of Agile Software Development [Coplien04]

Many of these tangle together a number of practices (and sometimes share) practices. For instance, Scrum is a practice that is already presented in a separated and re-useable fashion - as a management practice for the planning and execution of iterative projects. It is completely independent of the engineering and other practices that a team is going to use . This separation and the ability to mix and match Scrum with other practices is one of the reasons that it has proven so popular and been so widely adopted.

Compare this with how iterative project management has been traditionally presented in the Rational Unified Process [Kreutchen03] - as a tightly coupled, inseparable part of a much larger process that doesn't seem to be usable without the Unified Process lifecycle, use cases, components, or a strong focus on architecture. This has always made it almost impossible for customers to take the practices that they want without having to take all of the other practices as well.

Thus, there are many places where you can already find practices:

- There are many well-formed practices already available
- There are many practices embedded in existing processes
- There are many tacit practices experienced professionals communicate by coaching and example

A good place to look for well-formed practices is in the way that training is presented and processes rolled out. The normal way to train people is one practice at a time. This is also the way that most successful process roll-outs are undertaken.

Practices evolve from people doing their jobs and sharing their experiences. What we need is a better way to capture and share these practices, one that avoids "religion" and self-promotion, allowing each practice to stand on its own.

So in the future, instead of having experts contributing yet another process, they would contribute separate and distinct practices. Each practice will be complete and help project teams drive their project forward.

What Kinds of Practice Are There?

As you might expect there are practices to address all the different areas of software development and team working, including:

- Software Engineering Practices – such as practices for developing components, designing user interfaces, establishing an architecture, planning and assessing iterations, or estimating effort
- Social Engineering Practices – such as practices on teamwork, collaboration, or communication
- Organizational Practices – such as practices on project milestones, gateway reviews, or financial controls

In time, all of these practices will be made available plus many more. Figure 1 illustrates a selection of the practices we would expect to be available in the future. Some of the practices will be complementary, such as Scrum, pair programming and use cases. Others will be competing with each other, such as use cases and user stories.

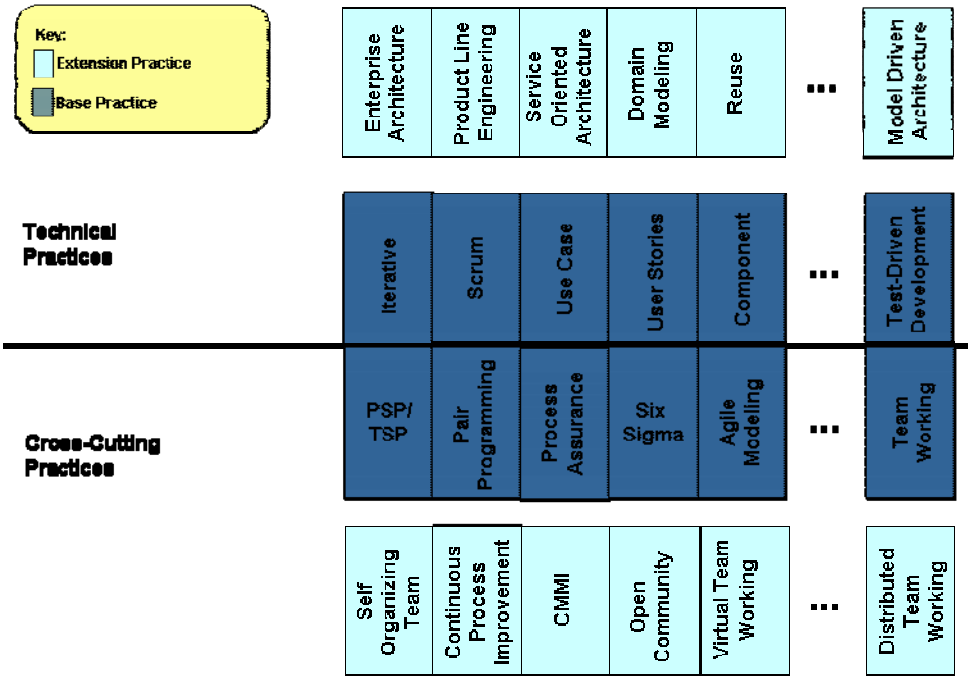


Figure 1: There are different types of practices

Figure 1 also shows that there are technical and cross-cutting practices. Technical practices deal directly with the production of the software or the other artifacts (such as requirements) needed to support the development of the software.

Cross-cutting practices are subtly different. Rather than focusing on the method you use to produce the software (such as iterations, use cases, and components), these practices address the softer side of software development (such as pair programming, team working, and communication). They don't directly describe how to produce software, but fundamentally affect how the team works and how the other practices are applied.

Many popular software development practices, especially the kind of social engineering practices promoted by the agile community, are cross-cutting in this way. The ability to capture and compose both technical and cross-cutting practices lets us address all aspects of software development in a simple, scalable and extensible fashion.

Figure 1 also shows a selection of peer practices and extension practices. The peer practices are full-fledged practices that can be applied separately and offer direct value to the teams and their customers.

Extension practices build on a peer practices to make them more scalable by addressing specific risks or adding complementary techniques. Teams will start by selecting the peer practices that they need to drive their work and add any other peer or extension practices as needed to scale up their way-of-working. The best kind of peer practices are those that focus on the essential elements of the practice, and leave the corner cases and specializations to be handled by other extending practices. This keeps the peer practices lightweight and agile, and prevents teams from inadvertently adopting practices that are more complex than they need to be.

The great thing is you only need to use the practices that you want. If you think the existing management practices are a waste of time, then you leave them out. By only drawing on the practices that you need, you can assemble the right way-of-working for your team, project and organization.

What Makes a Good Practice?

A well-formed practice addresses a specific aspect of software development or team working. A practice provides the guidance to characterize the problem, the strategy to solve the problem, and instructions to verify that the problem has indeed been addressed. It also describes what supporting evidence, if any, is needed and how to make the strategy work in real life.

Put simply, practices describe what to produce, how to produce it, and the competencies needed to perform the practice. Practices also provide pre-defined patterns to tailor and tune their use. The use of patterns allows practices to describe things such as appropriate work patterns (for example, collocated teams, distributed teams, or virtual teams), and ownership patterns (for example, common ownership or ownership by component). Applying the patterns easily allows a team to adapt the practices to their specific needs.

To be a well formed practice - one that can be applied safely and consistently – a practice must verify itself. So if a practice describes how to write code, it must also include how to test it. Or if a practice describes how to capture requirements, it must also

describe how to verify that the system produced meets these requirements. If the practices don't do this, then their ability to be applied successfully is compromised. This need for practices to include their own verification can be seen in the initial set of practices that we have developed to demonstrate and prove our approach.

If you examine the set of practices in Figure 2, you will see that there is no testing practice. This is because testing is an integral part of all the technical, practices. For example, the component practice takes a test-first approach, developing unit tests before the components are developed.

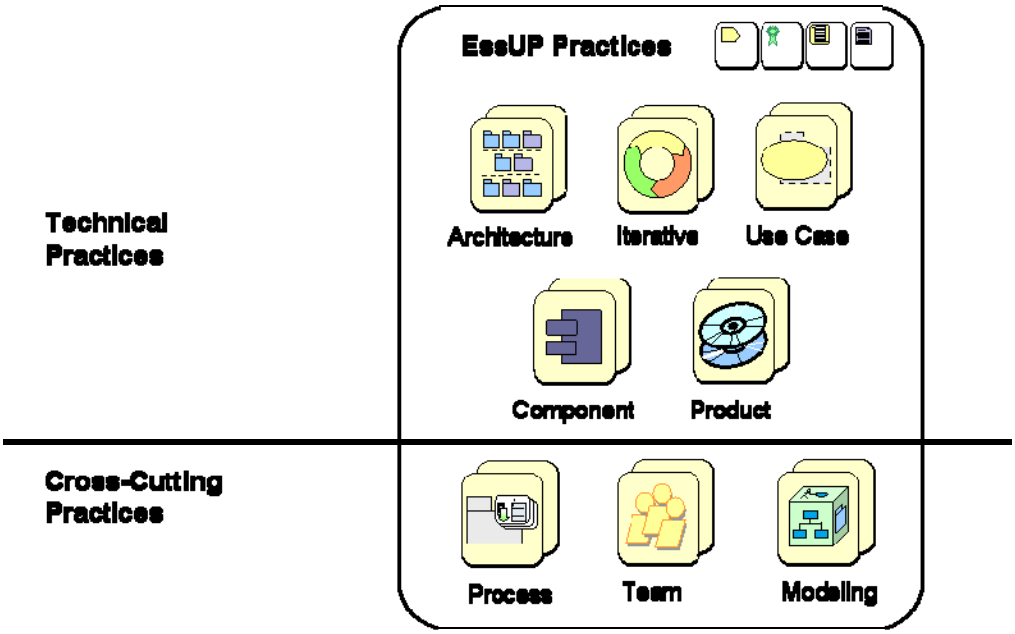


Figure 2: The Practices in the Essential Unified Process

Together, these eight practices form the Essential Unified Process (EssUP), a lightweight version of the Unified Process. EssUP includes the five technical practices found in all Unified Processes, and some new cross-cutting practices that draw on other areas of experience such as CMMI and Agile methods.

These practices can be applied separately or in concert with one another. Many teams are already using these practices. They all use different combinations and mix these practices with their own existing practices. Figure 3 shows how the EssUP practices can be mixed and matched with other practices to support different teams and objectives.

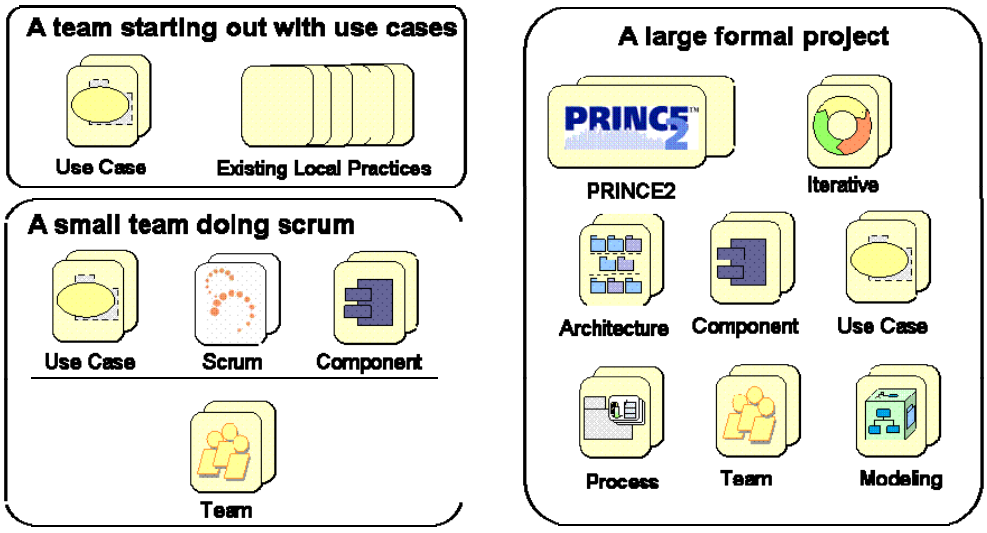


Figure 3: Mixing and matching the EssUP practices

The initial set of EssUP practices is designed to focus on the essentials of each practice – ensuring that they are lightweight and compatible with agile values and thinking. If more rigor, ceremony or documentation is required, then additional extension practices can be added. These extension practices build on the essentials to address specific problems and areas of risk.

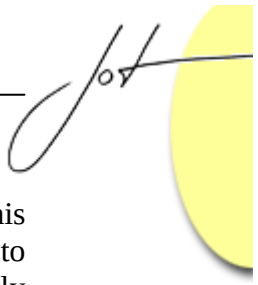
What Happens to "Processes"?

Every process can be considered a collection of typically tangled and tightly coupled practices. Once existing practices are separated, methodologists can focus on capturing best-practices in a reusable and extensible format without repeating or replacing existing practices.

Treating processes as collections of practices fundamentally changes the role of the processes. They just become a short-hand way of referring to a known set of mutually supportive practices, and their adoption acts as a useful starting point or goal for projects trying to change their way-of-working.

Instead of learning or adopting an entire process, practitioners will learn about individual practices and adopt these practices incrementally to improve their way-of-working. First, they select the most appropriate practices to address their needs and help them cope with the challenges of their current situation. Then, they adopt these practices in whatever combination and at whatever speed suits them. Most importantly, they add new practices to their existing way-of-working without having to change everything or throw-away the practices that they already know and love.

If they want to, teams will be able to start afresh with a totally new way-of-working, but experience has shown that it is much more effective to transform the way-of-working one practice at a time. When it comes to process improvement, a big-bang approach doesn't work. Trying to change everything, all at once, is a high risk strategy - one that has been demonstrated to fail time-after-time. Even if you want to move to a totally new



way-of-working, it is easiest and safest to do it one or two practices at a time. This minimizes the disruption caused by changing working practices, and provides a focus to the coaching needed to embed the new practices in the team. It also allows you to directly address your problem areas without having to change the practices that are already working well.

In the future, you will combine practices from many sources to create the way-of-working you need. Rather than talk about the process you follow, you will talk about the practices you use. If someone else tells you about a new practice, you will be able to try it without affecting the other practices that you are already using. You will even be able create and develop your own practices, then blend these with standard practices to create truly new and innovative ways-of-working. You will not be tied to any one process camp or ideology; you will be able to mix-and-match practices from any source to continuously improve and tune your way-of-working.

3 BRINGING PRACTICES TO LIFE

So, the time has come for processes to reinvent themselves as sets of separate but collaborating practices. For this reinvention to happen, new approaches are required to make the practices more accessible, to assemble them into a coherent way-of-working, and to let you apply them in the way that you want. Additional innovations are needed to make a practice-based approach work.

A New User Experience

Let's face it, most process and practice information is presented in a really boring way. It either comes as a book designed to be read from cover to cover, or as a web-site comprised of endless pages of scrolling text, most of which is of little relevance when actually engaged in developing software. Instead of presenting the practices as a book or a series of static html pages a new approach that focuses and contextualizes the information is required.

To change how people use process descriptions, we use a "card metaphor" (as in 5x3 inch index cards) to present the most important things about a practice. The cards, samples of which are shown in Figure 4, immediately bring the practice to life. They present a succinct summary of the most important things you need to remember about the practice. On their own, they provide enough information for you to apply the practice – including information such as where you are, when you can stop, and when you are finished. In most cases all you need to apply a practice is the set of cards.

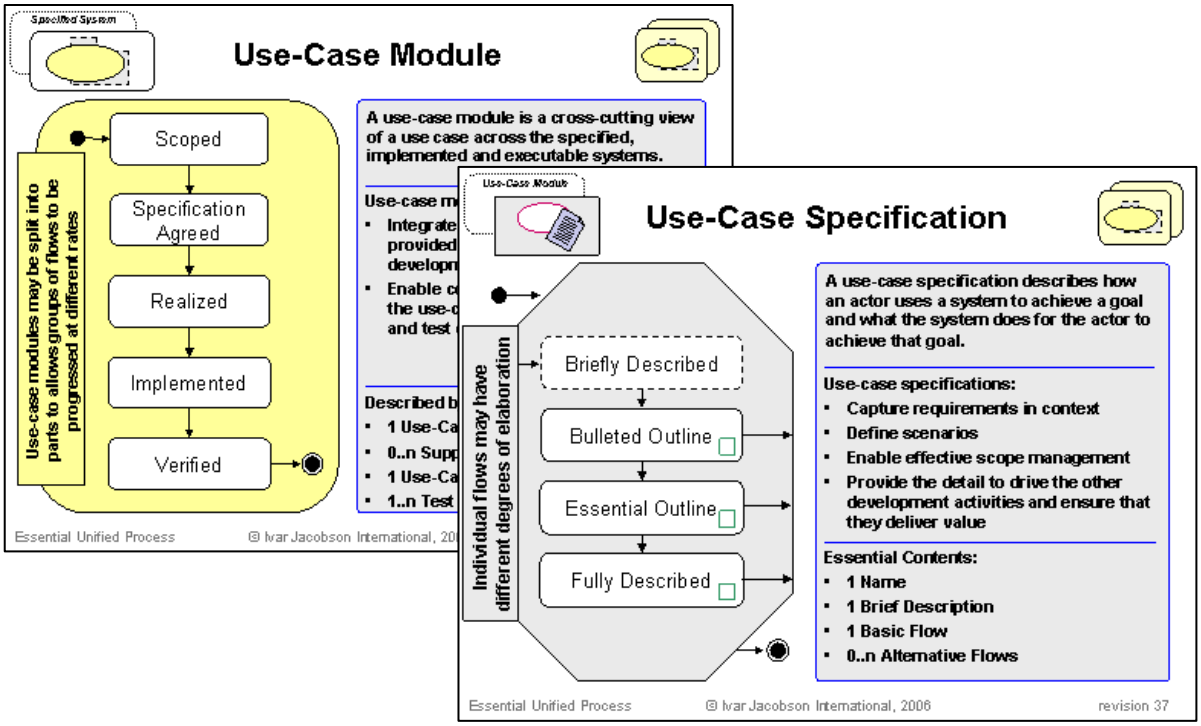


Figure 4 – Some physical cards from the Use-Case Essentials Practice

The cards also help you adapt the practice. You can scribble on the cards to make them more specific to your project needs. This provides a unique way to tune the practice on the fly and capture any lessons learned using it. You can even write out additional cards to extend the practice as come up with new ways of doing things.

Every card has a guideline presenting the next level of essential information to help you apply the practice. The guidelines are kept short (two – four pages) and to the point. Thus, they have more detail - but not too much. In teams, you expect members to have different backgrounds and competencies. Competent members use the cards to drive their work, while the guidelines put less experienced members on the same page. If team members are novices, no amount of textual descriptions will help. Consequently, we recommend they receive coaching from a more experienced team member, use an active practice, go for some training, or read a book.

In this way, practice descriptions are deliberately kept succinct and light weight. This is good because their goal is to focus on the essentials, which by definition are a sub-set of the entire practice area. Moreover, there is no need to repeat existing information (in books or papers, for instance). The intention is not to supplant or replace the existing reference material, but to complement it with a simple description of the practice in a form that can be used on a daily basis when developing software.

As shown in Figure 5, the guidelines refer to additional support materials. They cite standard references and information sources, rather than trying to re-write or replace them. This is particularly powerful when presenting your existing practices in this new

format, as you need only distill the essentials for your cards and guidelines, rather than reformating and re-writing all the existing information. Practices can now be presented as a set of printed cards and guidelines (and used in the way that XP projects use User Story Cards), and electronically as part of an active, integrated way-of-working.

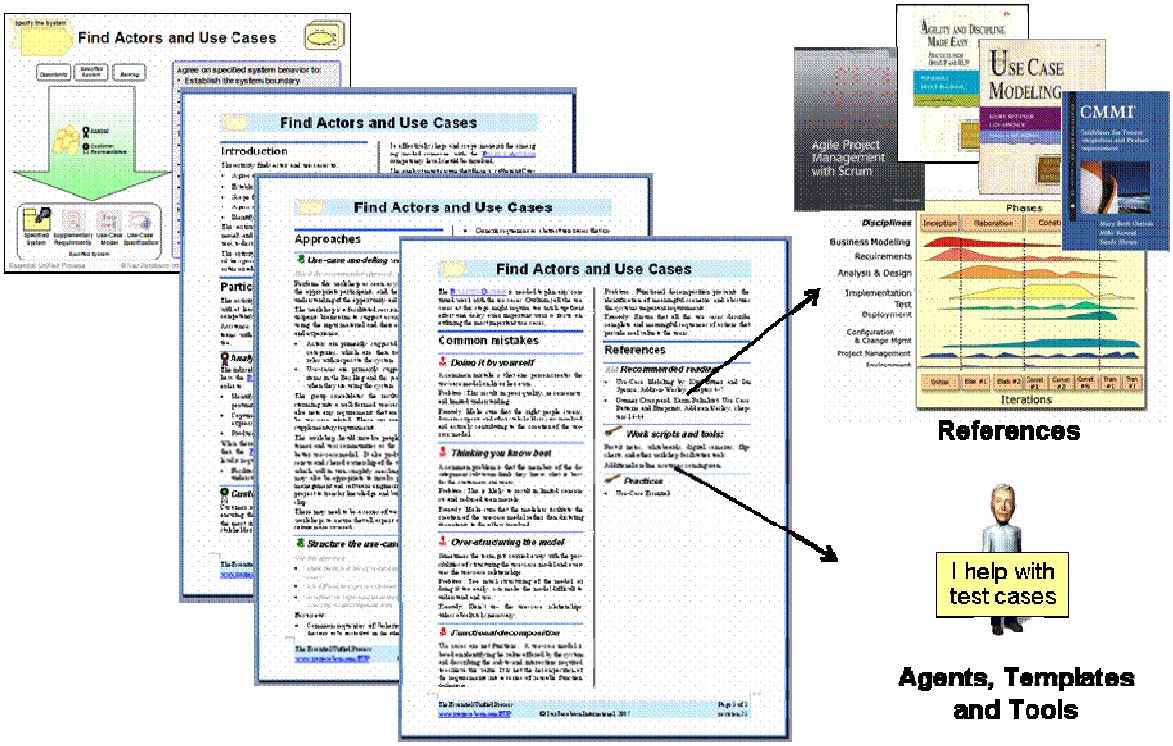


Figure 5: A card with its guideline and links to other references and tools

Having access to the physical cards and guidelines is very useful. You can use them during workshops and training events. You can distribute them as handy desk guides and crib sheets. Some teams even like to use the physical cards to organize and drive their work. The great thing about the physical cards is that you can easily manipulate them - write on them, group them, and even hand them out to the people that need them. Figure 6 shows some cards being used in this way to facilitate a workshop.

An electronic format is also needed. In the electronic environment the cards act as a view onto the practice providing direct access to the wider network of information and tools. They can also be annotated and managed in the same ways as the physical cards. Of course you can print the electronic cards as physical cards if you want to.

Figure 6: Using the cards to facilitate a workshop

The ability to manipulate and access the materials in both a physical and electronic fashion lets teams work in whatever way suits them. Most teams will use a mixture of physical cards (to facilitate team communication and group events) and an electronic environment (to provide on-line help and access to the way-of-working).

Assembling a Way-of-Working

Once practices have been separated, there needs to be some mechanism to compose the practices to form a team's way-of-working.

Step 1 - Start From a Practice-Independent Kernel

To enable the practices to be composed, we need a starting point - something that, although practice-independent, provides the basis for the definition and composition of practices. We call this starting point the software development kernel or, when using the cards and the card metaphor, the "software development game board". The kernel is a light-weight software development process, which, due to the absence of any concrete practices, is almost entirely tacit. Many of the underlying concepts driving modern software development practices are embodied in the kernel. This is not surprising, since all software development teams handle the same concepts and share the same mission - to develop good software. Having the practices share these common concepts is key to enabling them to be defined separately, yet be seamlessly composed together.

For example, the kernel contains the concept of the "specified system". Every project has to nave a shared understanding of what the system is supposed to do - the system's requirements or specification, as it were. This understanding can take many forms and be communicated in many ways, and the kernel doesn't care how this is done - it just reminds you that it has to be done. There are many practices you could use to specify the system: anything from having a quick conversation with the customer by the water cooler to producing a formal system requirement specification. It is up to the team to pick the best practice to meet its - and its customer's - needs.

The kernel also contains the concept of a "backlog", a central concept in Scrum and other management methods. Working from a backlog and prioritizing the work it contains ensures that work is not lost. The presence of the backlog in the kernel lets the kernel be

used to guide your software development, even when no practices have been selected. Again, just like specifying the system, how you address the work in the backlog is undefined and limited only by your creativity. Of course, there are practices to help you do this. The kernel provides the mechanism to link these practices together and focus the team on producing working software.

The kernel is the starting point for assembling a team's way-of-working. As shown in Figure 7, practices can then be added to the kernel to assemble the team's way-of-working and make it explicit.

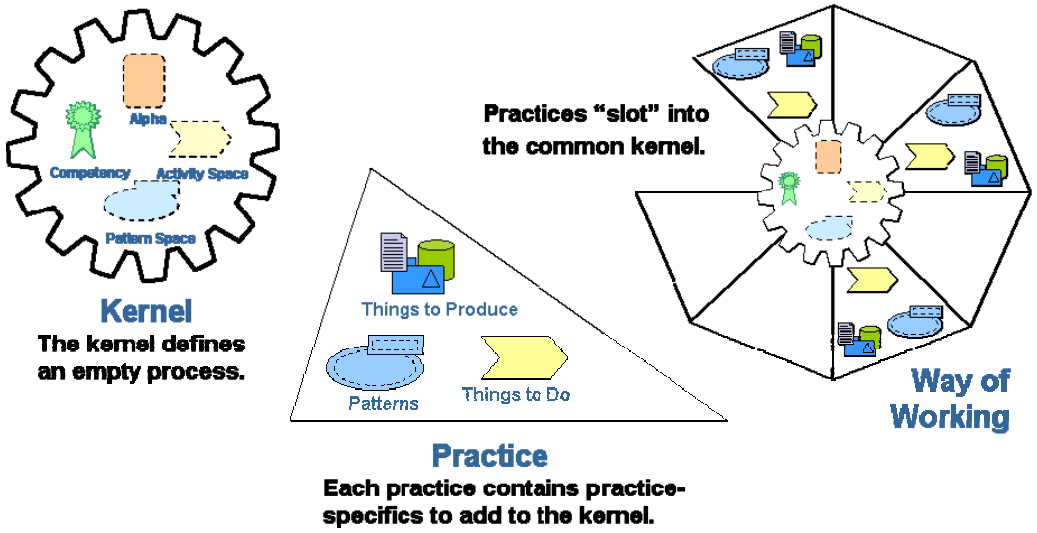


Figure 7 – Practices slot into the kernel to produce the composed way-of-working

Each practice brings its own approach to solving one of the problems of software development. For example, there are many ways of specifying the system to be built: you could use user stories, use cases, or declarative requirements. Each of these approaches would be expressed as a different practice and define different things to produce and different things to do. This is illustrated in Figure 8, which shows how a number of different practices can fill the same "holes" in the kernel (in this case, the kernel elements are "specified system" and "specify the system").

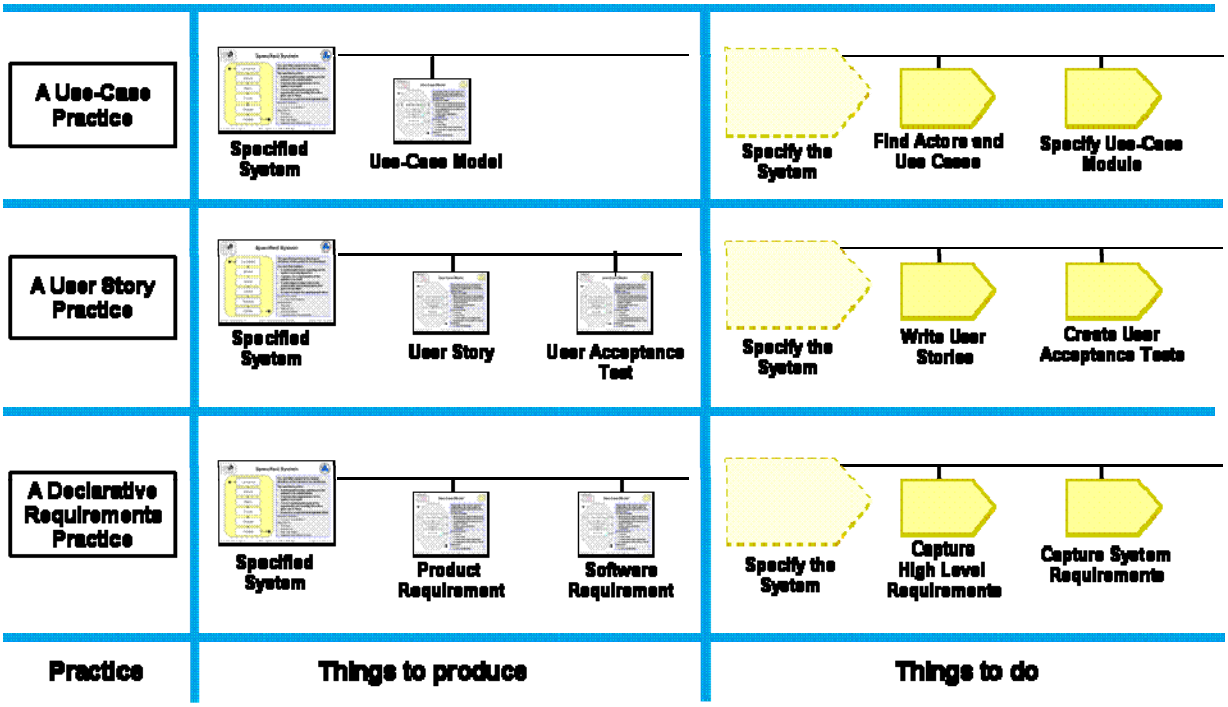


Figure 8: How different practices can achieve the same objectives

Step 2: Add Practices as You Need Them

At any point in time, the team can select a practice and compose it into their way-of-working. This will result in any existing cards and guidelines being augmented with the newly selected practices. The infrastructure will then adjust the tools and environment to reflect and support the new set of practices.

Using cards to help assemble and tailor the way-of-working is very effective. Cards from the various practices can be compared, arranged and annotated to provide a snapshot of the team's way-of-working. To compose the process from physical cards, you first identify the hole in the kernel you want to fill, and then add one or more practices to fill the hole. Figure 9 shows how different requirements practices can be mixed-and-matched to define a specific way to specify the system.

To add a practice to MyProcess simply drag it from the Practices area onto MyProcess, and then proceed with the composition.

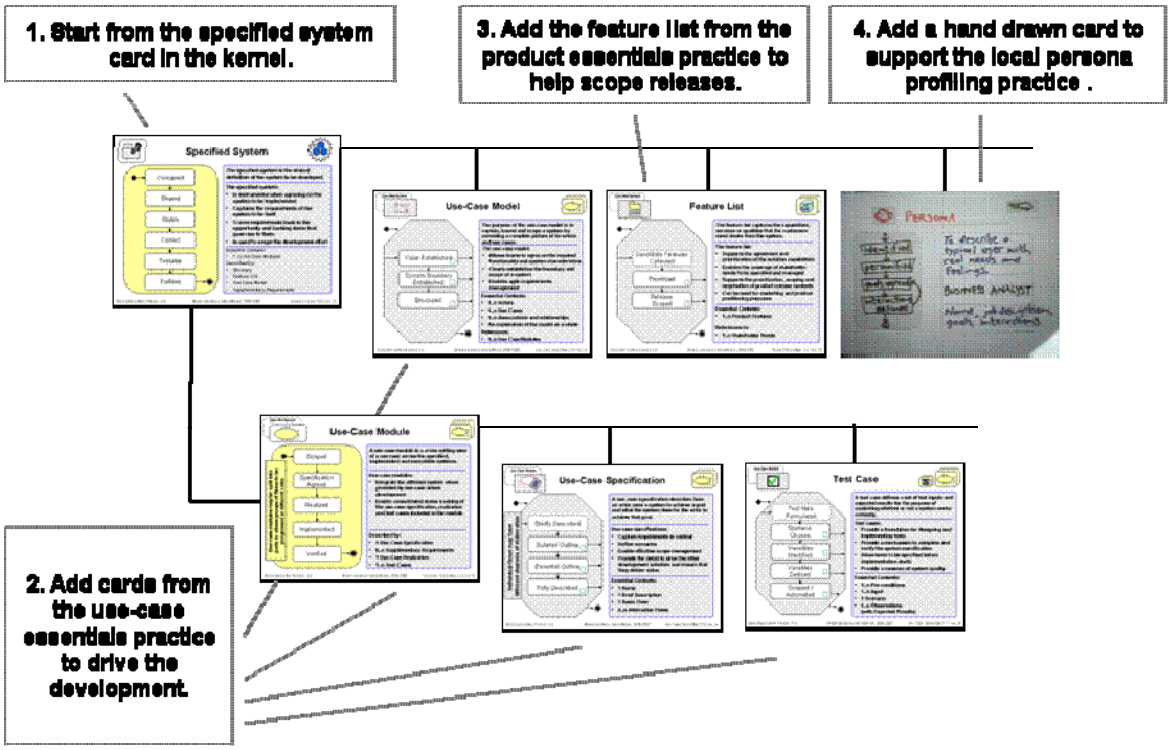


Figure 9: Assemble your Way-of-Working

Once the reasoning and negotiation has been completed, selected practices can be easily assembled within the electronic environment to capture the resulting way-of-working, and help the team apply the selected practices. Figure 10 shows how easy it is to manipulate and add new practices within EssWork.

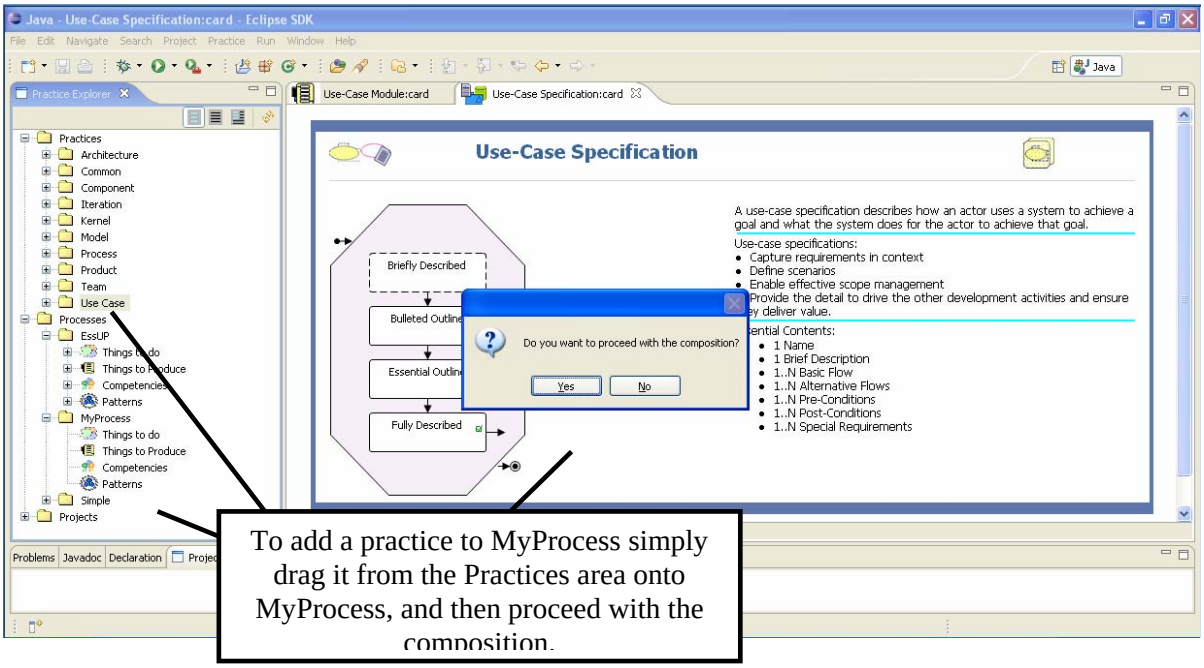


Figure 10: Dragging and dropping practices to create a way-of-working

Local practices can easily be integrated into the way-of-working by creating additional cards to represent their products. This lets us align the team's actual way-of-working to the set of cards in use, and keep the set of cards up to date. This provides a simple and easy to use mechanism to prevent the project and the process from getting out-of-sync.

A New Way of Using Practices

For practices to be useful, they need to be more than just something you read as a reminder or to help you understand an abstract concept (such as iterations). They must also help you understand your project and progress more effectively. We do this by further refining the card metaphor. Just as we have CRC (Class Responsibility and Collaborators) cards to identify and design classes, we use cards to facilitate almost every facet of software development.

This allows the cards to play a role throughout the project, rather than just being used when the team is learning or preparing to start work. With this in mind, we would like to highlight two ways to use the cards.

One way is when you work with the practices on the project. The cards make it easy to apply the practices and create tasks for the team to undertake. Among the most important things presented on the cards are the lifecycles of the things involved in the practice. These let the cards be used to visualize the current state of the project and determine the next move(s). You can do this as part of project planning or on the fly as the team decides what to do next. As you are always looking at the next move, you always know which parts of the selected practices to apply. If things have already been

done or are already in place, then practices to achieve these results are not needed and can be ignored.

Collections of cards can be assembled to represent the work to be done. By writing on the cards, we can capture exactly what's going on: estimates, progress to date, and who's doing what. We can then use the cards to build the backlog, the task lists and the iteration plans. Figure 11 shows an informal iteration plan that has been put together using the cards, post-it notes, colored pens, and a flip chart.

By relating the work to the practices in this way, the practice information is always immediately at hand and it is clear which practices are to be applied when. They can even be tracked by sorting and writing on the cards, or by entering the work items they represent into a spreadsheet or task management tool.

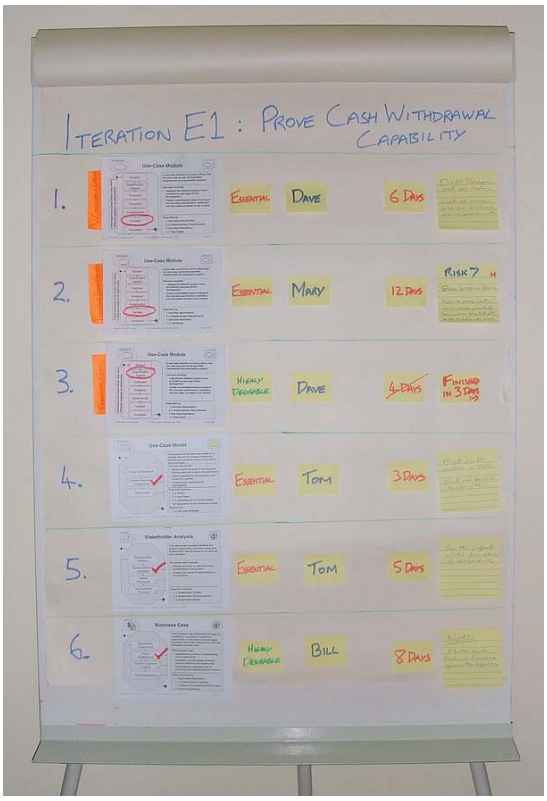


Figure 11: An iteration planned with physical cards

A second way to use the cards is when the team applies a single practice. One of the most exciting things about this approach is the ability to apply a single practice without changing the rest of the things being done on the project or re-working the things already produced. By using the kernel to drive the project forward from its current state to its desired state, you can apply the practices you need when you need them. This means that teams can select and experiment with new practices without having to undertake long and wasteful process engineering or documentation exercises.

The cards can also be useful when you plan and track the project and when you improve your way-of-working.

A New Way to Bring the Practices to Life

To really make the quantum leap, practices must not be passive, they need to be active. By active, we mean that the practices actually do things for the practitioner, something that is made possible with the advent of intelligent agents and smart tools.

Practices contain information which can be readily formulated as rules specified in a rule language. These rules can then be interpreted by intelligent agents and smart tools to help users apply the practice. This means that you can tell the practice to actually do things for you. For example you could ask an intelligent agent to retrieve the input information needed to perform a task, perform standard operations to carry out mundane tasks, orchestrate scripts based on your input, apply patterns and transformations, or run completeness and quality checks on your behalf.

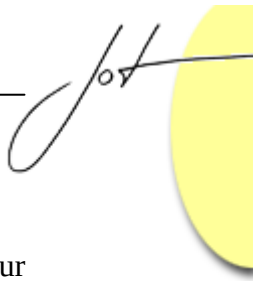
Intelligent agents can also dynamically configure the practice materials to provide the level of guidance and help you require. For example as a newbie you may want to have step-by-step help made available to walk you through an activity that you have never done before. When you are more experienced you may have evolved your own approach to tackling the piece of work and find this kind of guidance at the very best annoying if not down right patronizing and insulting. Intelligent agents can dynamically show you what you need, when you need it, according to your personal preferences, level of experience and project context. By delivering even more dynamic content intelligent agents can take the user experience to the next level and provide you with a user experience that suits you – one that integrates information from a number of sources including any content available through the Internet. For instance while you work on a new service for banking, an agent may be able to download example use-cases from other similar application for you.

Intelligent agents are not a new idea and have been in use for many years to support software development. For example WayPointer from Jaczone has been used to help teams with UML modeling and requirements capture. The results are very positive - Tata Consultancy Services have reported that analysts increase their productivity by more than 20%, the quality is substantially higher and the time for learning is considerably shorter when using WayPointer to support requirements analysis and use-case modeling. Most interestingly, the people getting help feel higher job satisfaction. Combining WayPointer support with the use-case essentials practice in this way provides the first smart EssWork practice.

The idea of activating a practice and making it smart is not about controlling what you do or making you work in some pre-defined way. It is about liberating you from the mundane and boring aspects of working in a software development team and allowing you to work creatively in the most effective and enjoyable manner possible.

Why waste time:

- Looking for things in the project repository?
- Performing and checking standard transformations?
- Clicking around monster intra-net sites for that really useful template or example?



- Preparing evidence for that review or assessment?
- Re-configuring those software development tools every time you change your way-of-working?
- Desk checking the code or model?
- Completing spreadsheets to prepare the information needed to drive the project measurements?

when an active practice could do it all for you?

The choice is always ultimately yours but the potential to make life on software development teams easier, more collaborative and more fun by using active rather than passive practices is there for everyone to see.

4 ESSWORK

While physical cards provide a good mechanism for understanding and applying the practices, they don't scale up for use by large projects or distributed teams – and can be a turn off for teams that are not enthusiastic about the cards or Agile approaches in general.

An electronic environment is needed to really let us compose the practices, generate the right cards and guidelines, manage the cards, activate the practices, and make the practices visible within the team's selected collaborative software development environment. We call this environment "EssWork", short for "Essential Work", and have implemented it to support the electronic version of the Essential Unified Process .

EssWork provides a practice-centric infrastructure into which you can load whatever practices you need (Figure 12). By default, the infrastructure includes the new card-based user experience, the practice persistence needed to store the team's way-of-working, and the interfaces for the development of practice adaptors to integrate EssWork into the team's development environment. It can also be complemented with practice activation technology (such as WayPointer) to bring the practices to life.

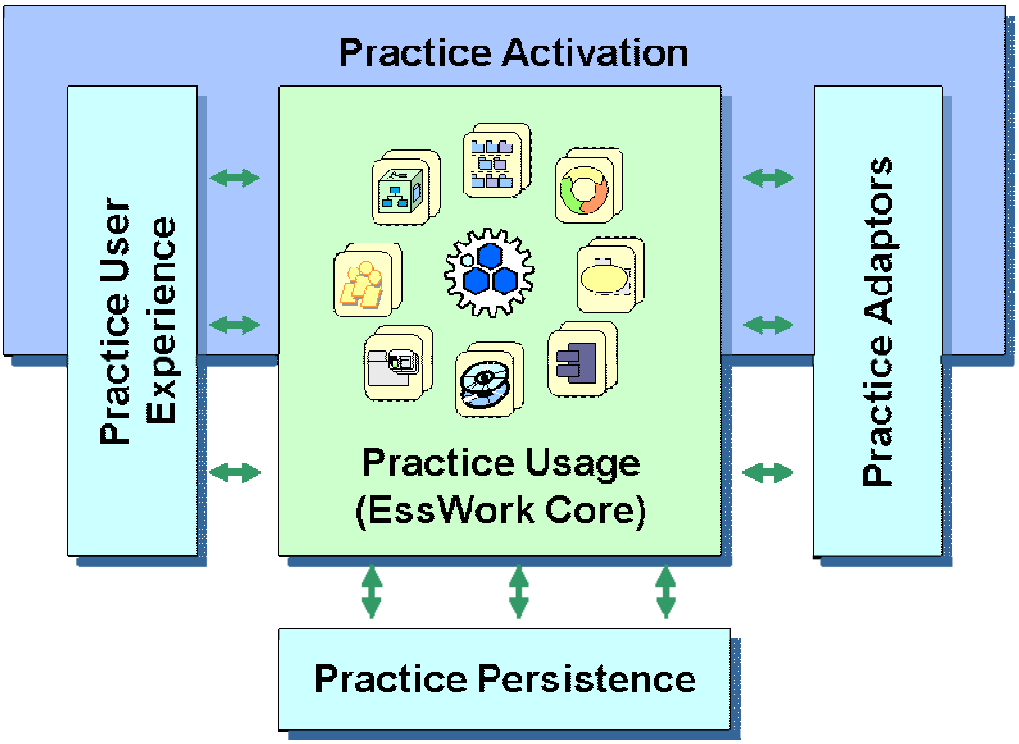


Figure 12 – EssWork Core and the Components of the EssWork Infrastructure

EssWork (www.esswork.com) is not a branded process. It is a practice-independent framework that provides the infrastructure and foundation for teams to compose their own ways-of-working. Software developers will not learn, adopt, or follow EssWork. They will learn, adopt and follow the practices that it brings to life. People won't say they are using EssWork when developing software any more than they say that they are using e-mail or word processors. It will just be a natural part of the team's infrastructure, enabling them to benefit from adopting and applying practices in an agile and disciplined fashion.

EssWork Core is freely available (www.ivarjacobson.com), and is to be donated, along with the technical practices from the Essential Unified Process, to the Eclipse Process Framework (www.eclipse.org/epf) where the EssWork community will be hosted. EssWork Core supports the use of practices over a rich set of client access platforms and collaborative development environments including Microsoft Visual Studio and Eclipse, and server side data stores such as Microsoft Team Foundation Server and JIRA.

5 TOMORROW IS HERE TODAY

We are on the threshold of a new era for software development processes. One where teams can easily assemble their own unique way-of-working by selecting proven practices, and mixing these with new and innovative practices that they invent for themselves. The foundations for the new era are:

- Practices are first class citizens

- Practices are presented in a simple, easy to use fashion
- Process becomes a composition of practices
- Infrastructure is needed to compose and execute the practices
- Practices can be made active

The foundations for the new era are already in place. Our suggestion to promote practices into first class citizens is not a mere proposal. As you have seen we have already developed a number of practices and an infrastructure for assembling and executing them. This means that the approach described here is already available today. You can already benefit from practice separation, and start to exploit the new infrastructure, by adopting a practice-centered approach using EssWork with EssUP. If you don't like the practices EssUP provides then you can easily develop your own practices and contribute them to the growing set of freely available EssWork practices.

Gone are the days when practices and processes are delivered as static web pages. The use of cards, especially when they are integrated directly into the project's way-of-working, breaks us out of the world of passive knowledge bases. No longer will we be faced with lengthy process descriptions that are divorced from what is actually happening on the project. Team's will be able to easily create and maintain a way-of-working that accurately describes what they do, and that helps them in their day-to-day business of developing software.

Intelligent agents allow us to go beyond this by removing the need for people to browse or search the process description. Instead intelligent agents enable the practices to interactively assist you while the work is being carried out, and reduce the amount of boring and repetitive work you need to perform.

Our work has been focused on making processes both agile and scalable. To do this we have developed several practices and an infrastructure to provide the practice composition and execution environment. The practices range from technical ones to those that deal with management and team collaboration. The infrastructure has been demonstrated to interoperate with Microsoft, IBM and open source technologies. While there is still a lot of work to be done, we are confident that we have established and validated a way to turn practices into truly first class citizens and an environment that will really make it worthwhile for developers to use the practices. We are ready for a new era, please join us there.

REFERENCES

[Jacobson99] Ivar Jacobson, Grady Booch, Jim Rumbaugh: Unified Software Development Process, Addison Wesley, 1999, ISBN 0-2015-7169-2

[Booch94] Grady Booch: Object-oriented Analysis and Design with Applications, 2nd ed., Benjamin Cummings, 1994, ISBN 0-8053-5340-2.

[Rumbaugh91] James Rumbaugh, Michael Blaha, William Premerlani: Object-Oriented Modeling and Design, Prentice Hall, 1991, ISBN 0-1363-0054-5

[Wirfs-Brock02] Rebecca Wirfs-Brock, Alan McKean: Object Design: Roles, Responsibilities and Collaborations, Addison-Wesley, 2002, ISBN 0-2013-7943-0

[Shlaer88] Sally Shlaer, Stephen Mellor: Object-Oriented Systems Analysis: Modeling the World in Data, Yourdon Press, 1988, ISBN 0-13-629023-X

[Constantine99] Larry L. Constantine, Lucy A.D. Lockwood: Software For Use: A Practical Guide to the Models and Methods of Usage-Centered Design, Addison-Wesley, 1999, ISBN 0-2019-2478-1

[Yourdon97] Ed Yourdon, Brian Henderson-Sellers, Ian Graham, and Houman Younessi: The OPEN Process Specification, Addison-Wesley, 1997, ISBN 0-2013-3133-0 .

[Palmer02] Steve Palmer, Mac Felsing: A Practical Guide to Feature Driven Development, Prentice Hall, 2002, ISBN 0-1306-7615-2

[Chrissis06] Mary Beth Chrissis, Mike Konrad, Sandy Shrum: CMMI: Guidelines for Process Integration and Product Improvement, Addison-Wesley, 2006, ISBN 0-3212-7967-0

[Emam97] Khaled El Emam (Editor): SPICE: Theory and Practice of Software Process Improvements and Capability Determination, IEEE Computer Society Press, 1997, ISBN 0-8186-7798-8

[Barney03] Matt Barney, Tom McCarty: The New Six Sigma: Leaders Guide to Achieving Rapid Business Improvement and Sustainable Results, Prentice Hall, 2003, ISBN 0-1310-1399-8

[Hunphrey97] Watts Humphrey: Introduction to the Personal Software Process, Addison-Wesley, 1997, ISBN 0-2015-4809-7 and Introduction to the Team Software Process, Addison-Wesley, 1999, ISBN 0-2014-7719-X

[Beck04] Kent Beck: Extreme Programming Explained: Embrace Change, Addison-Wesley, 2004, ISBN 0-201-61641-6

[Schwaber04] Ken Schwaber: Agile Project Management with Scrum, Microsoft Press, 2004, ISBN 0-7356-1993-X

[Cockburn04] Alistair Cockburn: Crystal Clear: A Human-Powered Methodology for Small Teams, Addison-Wesley Professional, 2004, ISBN 0-201-69947-8.

[Highsmith00] Jim Highsmith: Adaptive Software Development: A Collaborative Approach to Managing Complex Systems, Dorset House, 2000, ISBN 0-932633-40-4

[Coplien04] James Coplien, Neil Harrison: Organizational Patterns of Agile Software Development, Prentice Hall, 2004, ISBN 0-131-46740-8

[Kreutchen03] Philippe Kreutchen: The Rational Unified Process: An Introduction, Third Printing, Addison-Wesley, 2003, ISBN 0-321-19770-4

About the authors

Ivar Jacobson

Pan Wei Ng

Ian Spence