Software Process Improvement and Gaming using Essence: An Industrial Experience

Pan-Wei Ng
Ivar Jacobson International, Singapore
Email: panwei@ivarjacobson.com

Abstract—Software development teams are often ill-trained and ill-equipped to improve the way they develop software. As a result, process improvement is often left to those from without the development team, such as quality assurance staff, coaches, and consultants. This paper describes a novel process improvement approach that helps development teams to quickly describe their existing processes and to discover for themselves ways which they can improve. This not only gives teams a greater sense of ownership, but also lays the foundation for sustained improvement. In particular, we demonstrate how to kick start process improvement through a workshop. This workshop introduces an element of fun through the use of games and dice throwing. This allows participants highly engaged during the workshop, which helps them truly appreciate how recommended practices can work in their context, as well as side effects. Central to our approach is Essence, a method-independent software engineering kernel. Essence’s object-oriented and state-based representation provides building blocks for teams to describe existing processes and to enact them (human) simulation gaming. This allows teams to explore improvement areas in a self-directed manner, and to test how new practices work in their context. Our experience in conducting a process improvement and gaming workshop for an embedded product company demonstrates the viability of our approach.

Index Terms—software development, software process improvement, cards, simulation, gaming, agile, kernel, language, alpha, Essence, fault slip through

I. INTRODUCTION

While most developers can quickly get up to speed when it comes to software technologies (such as web, mobile, cloud technologies, etc.), it is much harder to raise capabilities in software development. In fact, software practitioners are very weak in software engineering knowledge such as requirements, analysis, architecture, testing, and project management [1]. Indeed, as Brooks [2] put it: “In no other discipline is the gulf between best practice and typical practice so wide.” Charette [3] in 2005 highlighted that the software industry spends billions of dollars entirely on preventable mistakes. Even today, IEEE runs a weekly “IT hiccups of the week” column with the 28 May 2013 issue [4] reporting that RBS (Royal Bank of Scotland) customers were unable to log into their accounts through their mobile phone app for 2.5 hours and Montreal’s Métro system shut down for the seventh time since July 2012.

Good software development teams are both competent in software technologies and software engineering. They are able to systematically pin-point where problems occur, find appropriate practices to resolve these problems, and fit the practices into their specific context. However, as mentioned, software teams are often ill-trained and ill-equipped to accomplish such improvements. As a result, process improvement is often left to those from without the development team, such as quality assurance staff, coaches, and consultants. This is not ideal because no one understands teams better than themselves. For lasting and sustained change, the development teams need the ability to self-improve.

To do this, a good and practical understanding of software engineering is critical. This is where Essence, a multi-dimensional method-independent software engineering kernel [5] plays an important role. Essence provides the building blocks through which teams can easily and quickly describe and model their existing process and discover for themselves how to improve. Essence achieves this in a lightweight and agile manner. This is in contrast to frameworks such as CMM, CMMI, ISO/IEC 15504:2004, ISO/IEC 12207:2004 etc. that Pettersson et al. [6] considers heavy weight and take much effort to put into effect.

Successful and sustained process improvement requires high team member participation levels and Dorling et al. [7] recommended using games to attain this. Interestingly, Essence’s object-oriented state-based representation provides a ready mechanism for simulating and gaming software development processes. Furthermore, this simulation and gaming can be conducted using the teams’ own processes with the teams’ own problems, which is very helpful for teams to understand how new practices work in their context.

In this paper, we demonstrate the value of Essence for process improvement and gaming. We used Essence as part of a process improvement workshop for an embedded device company. The software development division of this company has 500 software engineers and is considering agile adoption. 38 engineers with different roles participated in our two-day process improvement and (human) gaming workshop. Through this workshop,
they not only attained a good grasp of agile development concepts, but also achieved several important outcomes: (1) established a prioritized backlog of practices to be introduced, and (2) get some internal coaches trained. This is an important first step in a process improvement endeavor.

This paper describes the procedures of our workshop, our experiences conducting it and the participants’ evaluation. We emphasize human gaming and leave the computerization of the game as future work. What is important here is that we establish the viability of the gaming mechanics through this workshop.

This paper is organized as follows. Section 0 gives a brief overview of Essence, its key concepts and its use. Section 0 describes how we conducted the process improvement and (human) gaming workshop, our experience and the results. Section 0 briefly describes the participants’ evaluation, and finally Section 0 concludes the paper by comparing this approach with related works and provides suggestions for future work.

II. ESSENCE OVERVIEW

This section gives an overview of Essence. Essence originates from the SEMAT (Software Engineering Method And Theory) initiative [8] founded by Ivar Jacobson, Bertrand Meyer, and Richard Soley in 2009 to improve the state of software engineering practices, and to promote the sharing and relevance of findings and results across industry, research and education [9]. Essence is a language and kernel of software engineering useful for describing and enacting software engineering methods [10]. Though Essence is still at its infancy, work on Essence is rapidly growing, such as applying Essence in small and large development [11] [12], as a framework for software engineering education [13], and as a framework for systematically reporting empirical findings [14]. This paper adds to this body of knowledge by exemplifying its use in software process improvement.

A. Essence Alphas

At the heart of Essence is the concept of alphas, which is an object oriented approach to identify typical dimensions of software engineering challenges as objects. Essence kernel identifies alphas that are common to software development such as Opportunity, Stakeholders, Requirements, Software System, Work, Team and Way-of-Working (See Fig. 1).

Opportunity is about the value and rationale for having the software system. Stakeholders are those who are impacted by the software system. Talking to the right stakeholders (or stakeholder representatives) is critical to getting to the right set of requirements. Requirements are refined gradually across development. Software System encompasses both the architecture and quality of the software system. Work is the effort involved to build the software system. Team represents the people developing the system, who might have different background and experiences. Way of Working is how the Team (including stakeholder representatives) develops the software system including the principles for doing so.

To make sustainable and healthy progress, a development team needs to consider all alphas (i.e. dimensions) of software development. Most risks and project failures are due to paying little attention to one or more dimensions. To guide development teams evaluate progress and health as well as to detect risks early, each alpha is associated with a set of states.

For example, the Requirements alpha has the following defined states: Conceived, Bounded, Coherent, Acceptable, Addressed, and Fulfilled. The idea is to have requirements Bounded, before thinking if the contents of the requirements are Coherent. If a team attempts to make requirements Coherent before getting them Bounded (i.e. understanding the boundaries), they may risk wasting their effort if the requirements boundary changes. The Essence kernel provides detailed checklist of what each alpha, and what each state means. It is important to note that the use of states do not encourage waterfall thinking. A team can have different sets of requirements (instances), each residing at different states. The kernel can deal with large-scale software development, which has multiple sets of requirements (instances), and multiple software systems (instances).

As another example, consider the Stakeholders alpha (in the Essence specification) has the following states: Recognized, Represented, Involved, In Agreement, Satisfied for Deployment, and Satisfied in Use. As mentioned earlier, talking to the right people is critical for project success. Thus, it is important to recognize who are the right stakeholders early in a project, make sure they are adequately represented and involved.

Through the alpha states, a development team can quickly and systematically evaluate their progress and risks. In addition, through the alpha states, development teams can quickly describe and model the way they do development, which is the basis for finding where problems occur and how they can improve.

B. Essence Cards

The Essence kernel specification provides detailed checklist of each alpha and their states. Each alpha states and their checklist are presented as a deck of cards (poker-card size) as depicted in Fig. 2. The cards provide a quick reference of the kernel alphas, but also as an interactive and tangible way to model the way they conduct development. We usually get development teams to discuss and describe their development processes by
laying out the cards on the table and shifting them around [11][12].

C. Essence Language

In addition to alphas and alpha states, Essence has other kinds of language elements (such as Activity, Activity Space, Competency, Work Product, Resource and Pattern) to better describe practices precisely (see Fig. 3).

Work products (e.g. use case models and specifications) are the physical documents and deliverables that serve as evidence of progress. Work products describe alphas and their organization. One distinctive feature of Essence is that it also models level of details in work products. Work products can be described using different level of details. For example, well-understood requirements may be described to lesser level of details as compared to requirements that are either complex or easily misunderstood. Moreover, at the beginning of development, work products started off with lesser level of detail and become more detailed (if necessary) as development progresses.

Activities (e.g. finding actors and use cases) progresses alphas from one state to another, producing or updating work products along the way. Activity spaces are used to organize activities. To conduct an activity effectively, activity participants need to have pre-requisite competencies. Essence also models resources (e.g. case studies) and patterns to provide further guidance development teams.

The Essence kernel identifies several competencies, namely: Stakeholder Representation, Analysis, Development, Testing, Leadership and Management. Each competency has 5 levels, from Assists, Applies, Masters, Adapts to Innovates. The idea is that each practitioner has a number of competencies to different levels. Roles (such as a project lead, scrum master, etc.), which are modeled as patterns, would require some competency levels to perform their responsibilities well. This mapping is usually practice and organization specific.

Essence further provides extensibility mechanisms to add more guidance on top of the kernel. Figure 4 depicts the Essence language architecture. At the bottom is where constructs such as alphas, work products, are defined. On top of the language, the Essence kernel identifies a number of alphas, activity spaces and competencies that universal across all kinds of development. Using the Essence language and kernel, practices such as scrum, acceptance-test-driven development, product line engineering, can be defined. These practices can also include home grown ones. These practices can introduce new alpha, work products, etc. to provide a richer description of software development.

Each development team with its unique challenges will select appropriate practices to compose their development method. Essence applies the concept of separation of concerns, commonly used to design software, to the design of software development processes. This allows teams to improve the way they develop software by replacing an inappropriate practice with a more appropriate one. With Essence, practice can be described precisely and how they affect are team and the impending side effects are made explicit. This is fundamental to successful and sustainable process improvement.

III. ESSENCE FOR PROCESS IMPROVEMENT AND GAMING: AN INDUSTRIAL EXPERIENCE

In this section, we describe how we use Essence in a process improvement and gaming workshop to kick start a team’s process improvement effort. Fig. 5 outlines the workshop, which comprise two intertwining threads: a process improvement thread (assessment, walkthrough, planning) and a (human) gaming thread (pre-gaming, gaming, post-gaming).

The process improvement thread begins by assessing the participants’ current process and problems. From the gaming perspective, this step provides the inputs to tailor the simulation game to the workshop participants’ specific context. The next step in the process improvement thread walks through the participants’ process to explore ways of improvement and make
recommendations about what practices participants ought to use. This is in effect running the (human) simulation game. Gaming involves walking through the alpha states from first state to the last state for all relevant alphas. The final step reviews recommended practices based on how they contribute towards improving the participants’ process, and put these practices into a process improvement backlog that serve as the participants’ next steps after the workshop.

38 participants from an embedded product company joined one run of our workshop. These participants comprised a broad range of roles, from department heads to testers. The participants’ goal was to evaluate how agile development would work in their context and more importantly how they could improve their processes.

We broke the participants into four groups and each group had an even distribution of roles. The author was the primary facilitator of the workshop pairing with a member from the product development company for two important reasons: (1) to provide company specific inputs and support, (2) to become the company’s internal coach when introducing the practices after the workshop.

A. Assessment (Pre-Gaming)

The workshop began with a brief introduction of agile development concepts and a brief introduction of Essence. **Describe Existing Process** – The facilitators guided the participants to describe their existing process, specifically their software development lifecycle (SDLC) using Essence alpha states (See Table I).

<table>
<thead>
<tr>
<th>Alpha</th>
<th>Phase</th>
<th>Requirement</th>
<th>Stakeholders</th>
<th>Systems</th>
<th>Work</th>
<th>Team</th>
<th>Opportunity</th>
<th>Beta</th>
<th>Alpha</th>
<th>RC</th>
<th>RTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>KO</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stakeholders</td>
<td>ES</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Systems</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Work</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Opportunity</td>
<td>Alpha</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alpha</td>
<td>Beta</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alpha</td>
<td>RC</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alpha</td>
<td>RTM</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Working, Team, Opportunity, Software System, and Work (see Fig. 6).

![Figure 6. Votes for workshop outline](image_url)

**Identify Specific Problems** – Next, we asked the participants the types of problems they face during development, when they were usually detected and when they ought to have been detected. This borrows the concept of fault slip through (FST) technique [15] that is used to evaluate the effectiveness of testing processes.

For example, the participant had a problem of “having different understanding of requirements between customers and the team”. This problem was usually detected at Requirements state **Conceived**, and ought to have been detected at Requirements state **Bounded** when attempting to agree on Requirements scope. We tabulated the types of problems raised and plotted the number of occurrence of problem introduction and usual problem detection by Requirement states in Fig. 7.

![Figure 7. Occurrences of problem introduction and detection by requirements alpha states](image_url)

From Fig. 7, it is clear that it would be beneficial for the participants to be equipped with the ability to detect problems earlier, especially when they were at the Requirements Coherent state. This phenomenon hints that a practice like Acceptance Test Driven Development (ATDD) would be very useful. By walking through the identified problems, the facilitator could also be able to determine what other practices that might be of useful to the participants. These practices would be exercised during walkthrough/gaming. To make the walkthrough/gaming interesting, we inject the same problems into the simulation to get participants discuss how they normally respond to such problems, and the impact.

B. Walkthrough (Gaming)

The second phase of the workshop involved the process walkthrough and gaming via the alpha states. This represents the most complicated part of the
workshop and this requires experienced facilitators who understand how to interact with the participants and address their specific problems through relevant practices. Table II summarizes the walkthrough (gaming) steps briefly.

**TABLE II. HUMAN WALKTHROUGH (GAMING) STEPS**

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Prepare List of Requirement-Items in a Backlog</td>
</tr>
<tr>
<td>2.</td>
<td>Plan-Do-Check-Adapt gaming cycle</td>
</tr>
<tr>
<td>a.</td>
<td>Agree on cycle objectives based on:</td>
</tr>
<tr>
<td>i.</td>
<td>Target alpha states (according to process)</td>
</tr>
<tr>
<td>ii.</td>
<td>Requirement-items to deliver</td>
</tr>
<tr>
<td>b.</td>
<td>Determine tasks to achieve cycle objectives</td>
</tr>
<tr>
<td>c.</td>
<td>Simulate cycle by throwing a dice for each task</td>
</tr>
<tr>
<td>d.</td>
<td>If 6 were thrown, inject a problem (from the identified list), otherwise task is completed.</td>
</tr>
<tr>
<td>e.</td>
<td>Discuss practices to resolve the problem</td>
</tr>
<tr>
<td>f.</td>
<td>Review development state</td>
</tr>
</tbody>
</table>

Preparing Requirement-Item Backlog – Step 1 in TABLE II is about creating a list of exemplary requirement-items that would be used to drive the walkthrough/gaming. This was not easy as time was limited in this 2-day workshop. Here the in-house facilitator’s help becomes very useful by providing company specific inputs.

Plan-Do-Check-Adapt Gaming Cycle – This was where the bulk of the walkthrough and gaming occurred. This was also the most complicated because the facilitator must balance between gaming realism and the lessons to learn for the participants.

Agree Cycle Objectives – At the beginning of the cycle (2a), the groups agreed on the cycle objectives, which was about choosing target alpha states and requirement-items that should be achieved within the cycle. The described process in Section 3.1 provided some guidance on the target alpha states to select. This cycle could coincide with an iteration, but generally speaking, setting cycle objectives can work without adopting iterative development.

Determine Tasks – Once the cycle objectives were agreed, the participants determined the tasks that could achieve the objectives. The estimated task effort had to fit within the cycle capacity (i.e. man-days).

Simulate Cycle – For each task in the cycle, participants would throw a dice. If a “6” were thrown, the facilitator gave the group a problem/impediment (from the ones identified earlier in Section 0.A).

Discuss Practices – For each problem/impediment, the group would discuss how best to resolve the problem, which normally involves some new practices such as the above-mentioned ATDD. This gave the facilitator a golden opportunity to discuss its use in the participants’ context. Implementing these practices would also introduce additional tasks. The ability to select and discuss appropriate practices and recommendations placed huge demands on the facilitator’s knowledge. This could be addressed by artificially limiting the kinds of problems the participants faced.

Review Development State – At the end of each cycle, each group reviewed whether they achieve the target alpha states. In our workshop, we also simulated the steps for iteration planning, iteration review and dialing meetings. The Plan-Do-Check-Adapt (PDCA) [12] cycles was repeated until the target alpha states as per the described lifecycle had been achieved, or when the participants understood how the recommended practices (e.g. ATDD) work within their context.

C. Improvement Planning (Post Gaming)

From the simulation gaming both participants and facilitators had good understanding of the problems and recommended practices. The participants evaluated the contribution of practices recommended during walkthrough/simulation phase described in Section 0.B.

![Figure 8. Contribution of practices towards software process improvement areas by alphas](image)

We counted the number of participants who believed that a practice had a strong contribution towards each improvement area by alphas and plotted the results (See Fig. 8).

Based on the results in Fig. 8, the participants wished to start introducing iterative development, and daily meetings to their way of working, followed by ATDD and cross-functional teams. These practices became part of their process improvement backlog (see Fig. 5).

In contrast to passively being told which practices to adopt, the walkthrough (gaming) had given the participants a firsthand experience applying practices in their context, albeit a simulated one. This would be useful when they actually adopt the recommended practices in their actual environment.

IV. WORKSHOP EVALUATION

We evaluated the realism and effectiveness of our workshop and asked the participants what they liked about the workshop and what could be improved, whose results are listed below:

What participants liked – Participants liked the interactive, collaborative and gaming nature of the workshop, and in particular gaming through their process. In fact, the participants had lots of fun throwing dice, which help them get engaged. This helped them
appreciate agile development concepts. They also appreciated the use of Essence alpha states that provided a systematic approach to think about progress and to describe their SDLC (software development lifecycle).

What could be improved – Although we were using a simplified version of their process, they preferred using a gaming example closer to their domain. In our case, this was not easy as we had participants from different departments building different kinds of systems.

V. CONCLUSIONS AND FUTURE WORK

Existing literature has no short of Software Process Improvement (SPI) frameworks. Pettersson et al. [6] highlighted that many of these frameworks use a heavy-weight one-size-fits-all approach towards assessment and as to what the organization needs.

Our approach is an example of an alternative lightweight approach, but with some important differentiators. Firstly, we conducted the assessment as part of an agile process improvement workshop. This meant that participants were able to learn something (i.e. about agile development) as well, as opposed to merely providing information by assessors. Secondly, the use of gaming simulations and walkthroughs help participants explore problems and solutions in greater detail. This adds to the first point and keeps participants constantly engaged. Thirdly, we used Essence as a common language in both assessment and gaming. Essence acts as a bridge between participants and facilitators. The participants described their processes and problems using Essence and the facilitator explained and recommended practices using Essence language. Fourthly, the workshop laid the foundation for further analysis and improvement. For example, the problem identification approach can be supplemented by analyzing historical records to measure frequency of problem occurrences.

A natural follow up to this work is to provide more guidelines so that less experienced coaches can act as facilitators. We would also like to try out different variations as to how we can conduct the workshop better. Computer-based gamification is an area that is receiving increasing interest [16], and naturally, we would explore this area further.

REFERENCES


Pan-Wei Ng received a Ph.D. in computer engineering from Nanyang Technological University, Singapore in 2002. He is the Asia-Pacific CTO of Ivar Jacobson International. He coaches and advises large organizations in Asia Pacific. He is the co-author “Aspect Oriented Software Development with Use Cases” and “The Essence of Software Engineering: Applying the SEMAT Kernel”.