HOW "DEVELOPER STORIES" IMPROVES ARCHITECTURE
Facilitating Knowledge Sharing and Embodiment, and Making Architectural Changes Visible

Rolf Njor Jensen, Niels Platz and Gitte Tjørnehøj
Institute of Computer Science, Aalborg University, Fredrik Bajersvej 7E, Aalborg, Denmark

Keywords: eXtreme Programming, Architecture, Design, Agile Development, Methodology, Practice, Systems Development, Software Quality, Developer Stories.

Abstract: Within the field of Software Engineering emergence of agile methods has been a hot topic since the late 90s. eXtreme Programming (XP) (Beck, 1999) was one of the first agile methods and is one of the most well-known. However research has pointed to weaknesses in XP regarding supporting development of viable architectures. To strengthen XP in this regard a new practice: Developer Stories (Jensen et al., 2006) was introduced last year mainly based on a theoretical argumentation.
This paper reports from extensive experimentation with, and elaboration of the new practice. Results from this experimentation shows that using Developer Stories increases the likelihood of developing a viable architecture through a series of deliberate choices, through creating disciplined and recurring activities that: 1) Facilitate sharing and embodying of knowledge about architectural issues, and 2) heighten visibility of refactorings for both customers and developers.

1 INTRODUCTION

Agile methods has been a hot topic since the late 90's (Beck et al., 2001), shifting system developments focus from processes and tools to individuals and interactions, from documentation to working software, from contract negotiation to customer collaboration, from following a plan to responding to change. eXtreme Programming (Beck, 1999; Beck, 2000; Beck, 2004) was one of the first methods within this paradigm and has become very popular within both research and industry. However XP has also been critiqued for not supporting the development of a viable architecture and hereby jeopardizing the quality of the developed systems.

What is design and what is architecture? No clear distinction can be made between the two in general, and in the case of XP, the two terms are frequently overlapping - i.e. both in terms of intention and scope. Architectural issues are dealt with through Enough Design Up Front combined with constantly designing and refactoring entailing that the architecture keeps improving but only on demand. The prevalent principle presented by XP to guide development of the architecture is "Simplicity", subject to the criteria: "Appropriate for the intended audience", "Communicative", "Factored" and "Minimal".

Does XP deliver a viable architecture? Embracing the values of XP, and following the practices in a disciplined manner in accordance with the principles, will lead to a sufficient and viable architecture (Beck, 2004).

However, discourses in literature and conference proceedings show that supporting development of a viable architecture still is a subject in XP. Some examples of the discourses are:

- The "Metaphor" practice that in the first book on XP (Beck, 1999) explicitly guided architecture, points to the need of focusing on a shared understanding of the system. The practice was found difficult to operationalize (Fowler, 2001) and was therefore removed (Beck, 2004), but soon it was missed (Lippert et al., 2003; West and Solano, 2005) and hints for operationalization was suggested.
- Several have proposed an introduction of some

Njor Jensen R., Platz N. and Tjørnehøj G. (2007).
HOW "DEVELOPER STORIES" IMPROVES ARCHITECTURE - Facilitating Knowledge Sharing and Embodiment, and Making Architectural Changes Visible.
In *Proceedings of the Second International Conference on Software and Data Technologies - SE*, pages 56-64
DOI: 10.5220/0001338900560064

kind of requirements management into XP (Eberlein and do Prato Leite, 2002; Paetsch et al., 2003) when quality of the system under development is a concern.

- Reports of problems trying to integrate large-scale refactorings into the everyday work of XP projects (Lippert, 2004). The quote "*... aggressive refactoring probably will remain the most difficult, because it requires consistency, energy, and courage, and no mechanisms in the methodology reinforce it.*" (Cockburn, 2006), brings the deficiency of XP into focus, offering an explanation of the many attempts at amending XP.

Developer Stories – a new agile practice – was introduced to address the deficiency of XP concerning architecture and design (Jensen et al., 2006). The new practice was inspired by the explicit focus on architecture in more traditional methods, but designed to fit perfectly in tune with the practices, principles and values of XP. Developer Stories manage non-functional and other architectural requirements for systems - in parallel with the feature-oriented user stories (effectively addressing functional requirements). The proposed practice was argued through a literature-based analysis (Jensen et al., 2006).

Investigating how Developer Stories works in a concrete development project is the subject of this paper. Analyzing Developer Stories, we propose a hypothesis: Developer Stories essentially contributes to the development process by two means: 1) facilitating sharing and embodiment of knowledge concerning architectural issues, and 2) improving visibility of architectural changes – thereby giving the customer more leverage to steer the development process. This paper presents the results of an experiment with Developer Stories, which sets out to validate this hypothesis, providing evidence that these two means are in fact present. The experiment is designed as a combination of a field- and laboratory experiment, and evolves around an XP project of 3 iterations with 6 developers, a coach and an on-site customer. Concluding upon the experiment, we find that our hypothesis is true, and that Developer Stories improve knowledge sharing and heighten visibility by creating reoccurring, disciplined activities for exploring and processing possible architectural issues.

The remainder of this paper is organized as follows: Section 2 presents the new practice Developer Stories in some detail (more arguments can be seen in (Jensen et al., 2006)). Section 3 presents the experiment settings, data collection and analysis, while section 4 lists the findings from the experiment. Results are summarized in section 5, and then discussed in section 6. Finally we draw our conclusions in section 7, and hint possible future work in section 8.

2 DEVELOPER STORIES

In the following we present in some detail the practice Developer Stories based on the original description of Developer Stories (Jensen et al., 2006). For practical purposes some aspects were clarified before the experiment, but in all important aspects the practice has remained as the former proposition.

The overall goal of Developer Stories is to provide the development team with the opportunity and means to improve the architecture of systems in development, accomplishing greater business value, and a viable architecture.

The Developer Story practice is designed to fit into the synergetic mesh of XP's practices. It takes its place in the symbiotical relationships of the practices, relating to i.e. pair programming, incremental design, test-first programming and user stories (for clarification of this see (Jensen et al., 2006)), It is designed to both support and be supported by the values of XP, and in general follow the look and feel of XP.

2.1 The Artifact: A Developer Story

Developer Stories as a practice consists of a process – a number of activities, and artifacts. The artifacts are the developer stories, defined as such:

> *The developer stories describe (changes to) units of developer-visible properties of the software. In contrast, user stories describe units of user-visible functionality of the software. The physical representation of a developer story is an index card, which may have another color than user stories, making it easy to distinguish them from each other.* (Jensen et al., 2006)

In essence, a developer story is much like a user story - but where a user story describes features of the system and is written by the user, a developer story describes changes to the system that are often not visible to the user, but highly visible to the developers – and is therefore written by the developers.

Figure 1 depicts a developer story from the conducted experiment. The layout of a developer story is an index card, and the form of the content is not bound by any formalism. The extent of the content is at the leisure of the developers. Whether it should contain a description of the problem, solution, tests, tasks and estimate, or whether it (as is the case of our example from the experiment, Figure 1) only needs two lines of description.

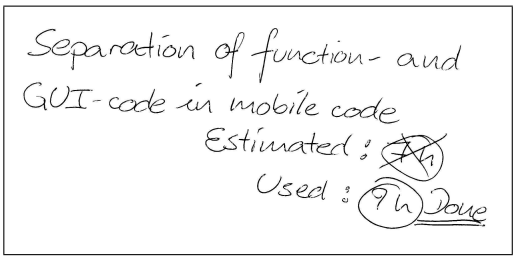

Figure 1: Example of a developer story.

2.2 The process

The activities that constitute the rest of the practice is interwoven into the other activities of development work, especially the discovery of potential developer stories and implementation of the chosen stories. However the exploration of the architectural issues, the writing of the stories and the costumer choosing which to implement is organized as an autonomous activity; The Architectural Game.

2.2.1 Acknowledging Architectural Issues

Integrated in the day-to-day activities of an iteration the developers acknowledge architectural issues that they experience. They either make a mental note, or preserve some artifact – a drawing on a flip-chart, a sticky-note or otherwise. Acknowledgement happens during pair-programming, shared discussions, lunch-breaks, etc., and serve as inspiration or reminders at the following Architectural Game.

2.2.2 The Architectural Game

The Architectural Game (see Figure 2) is a two-hour meeting that occurs once every iteration just before or after the weekly planning meeting (Beck, 2004) (formerly known as the planning game (Beck, 2000)) and may well be performed as a stand-up activity. The purpose of the game is to explicate and visualize refactoring needs as stories.

As shown in Figure 2 the game is based on experiences (including the notes and artifacts mentioned above) from the past iteration(s). First the developers collaboratively explore and elaborate architectural issues and eventually write developer stories expressing refactoring needs. Then they estimate the stories and may prioritize them. The customer is optimally present at this part of the Architectural Game as bystander. The participants shift rapidly, fluent and imperceptibly between activities.

When the developers have expressed the refactoring needs that they have found in terms of developer

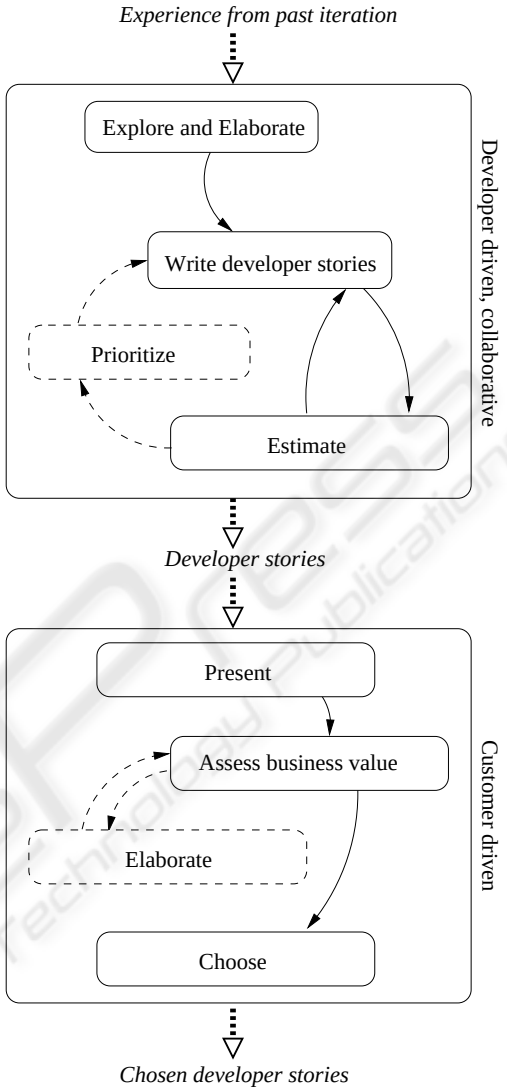

Figure 2: The Architectural Game.

stories, the stories are presented to the customer to choose from. He is responsible for gaining insight in the developer stories, so that he can assess the business value of the stories, based on his detailed business knowledge. This might require more elaboration or further explanations in non-technical terms by the developers.

2.2.3 Implementing Developer Stories

The chosen developer stories are handled similarly to the chosen user stories during the following iteration, e.g. developer stories are implemented and tested analogously to user stories. New unit tests are added and existing unit tests modified. Just like acceptance

tests for user stories, acceptance tests for developer stories are written prior to implementation by the customer, though supported by the developers – to uphold the double checking principle (Beck, 2004).

3 EXPERIMENT

The overall aim of the performed experiment was to gain insight into the workings of Developer Stories, to understand how and why they possibly accommodate architectural improvement.

3.1 Hypothesis

We regard the architecture of a system, as something that at any given time can change in several different ways. Choosing which changes to implement, is a matter of assessing which changes (if any) that will provide the largest increase in the value of the system. But how do we determine which changes are possible and which are feasible? And when a set of potential changes is identified, who then chooses which to implement? These two questions forms the basis from which our hypothesis is formed.

Finding Feasible Changes Creating the architecture of a system can involve many different stakeholders, but it is the developers that have detailed knowledge of the design and architecture of the implementation. With this in mind, the different developers are all potential changers of architecture, and as such the root to uniformity or chaos. Fowler promotes an unorthodox definition of architecture:

> *In most successful software projects, the expert developers working on that project have a shared understanding of the system design. This shared understanding is called architecture.*(Fowler, 2003)

Following this definition, and the previously stated premises, it is necessary to build and maintain a shared, uniform understanding of the existing architecture, in order to identify which changes to the architecture may feasibly be implemented. Hereby it becomes imperative that any practice aiming to improve architecture facilitates knowledge sharing and embodiment.

Choosing Which Changes to Implement While the developers (may) have an understanding of the architecture and possible changes, they do not posses all knowledge required to determine the change of value of the system entailed by effectuating a change. The missing knowledge is possessed by other stakeholders, which in XP are represented by the on-site customer. The required knowledge is i.a. the context in which the system is to be deployed, the context to which the system is developed, etc. If this knowledge is to be employed, the customer must be in a position to execute influence on which changes are implemented, and it follows that a practice aiming to improve the architecture must provide the customer with visibility of possible architectural changes.

In short, our hypothesis is that Developer Stories contribute to the development process with knowledge sharing and embodiment of architectural issues among the developers, and gives the customer visibility of the possible architectural changes.

3.2 Experiment Setting

The experiment was a full scale XP-development project integrated with the Developer Story practice. It was conducted in a laboratory but striving to share as many aspects as possible with a field experiment (Galliers and Land, 1987).

The field aspects were: 1) The laboratory was physically set up as a real XP room. 2) 11 out of 13 primary practices, and some secondary practices were employed in the software development process. 3) The development task was an industry request and the on-site customer was employed at the requesting company. 4) There were six developers, which is a normal size project. 5) We followed three one-week iterations, which is adequate embody both ordinary as well as the new practice of XP and thus to study the effect. 6) The implemented software was real-world: a rather complex client- and server system applying new technology used to scan barcodes on parcels in a freight registration system for a truck fleet operator. 7) After the experiment the IT-system was actually implemented at the requesting firm.

The laboratory aspects of the experiment were: 1) The developers were fifth semester computer science students with some experience in developing systems, but unskilled in XP. 2) The researchers played the role of coach and one on-site customer and the project was conducted at the university – not in an industry setting.

3.3 Preparing the Experiment

Preparation for the experiment was done through three activities: Teaching the to-be XP developers XP, arranging for a laboratory in which to conduct devel-

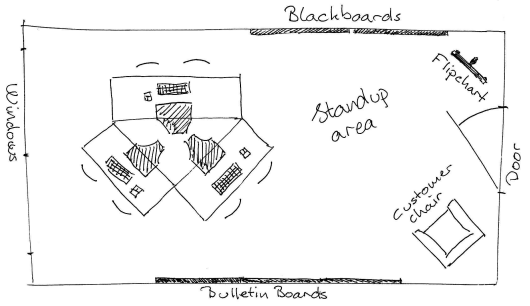

Figure 3: Configuration of the development room.

opment, and finally giving the developers insight into the product domain.

The developers were taught XP from scratch during a 20 hour course over a period of 5 days starting with and emphasizing the values of XP, later adding the principles, practices, and other features of XP.

The laboratory was set up as sketched in Figure 3 with one end occupied by three pair programming stations facing each other and the other end empty except from the information radiators – blackboards, bulletin boards (storywall, etc.), and a flip chart. This area also served as stand-up area. Finally a chair for the on-site customer was added. Compared to the recommended setup (Beck, 2004), denoted as "caves and common" (Cockburn, 2006) there is no private space for the developers to do solitary work. The need for caves was met by another room just down the corridor in which the developers also worked off-project hours.

3.4 Executing the Experiment

The experiment was conducted from mid-October to end-November. In total, three one-week iterations where completed. Development time was scattered throughout this period, concentrating more and more towards the end. The customer in cooperation with the researchers wrote the user stories for the project. The developers were enthusiastic about both XP and the product. The first iteration was a learning iteration with heavy involvement of the coach, but already in the second iteration the team, to a large degree, worked independently of the coach. The project was halted after the third iteration as planned only because the students had to refocus on other parts of their study. The product was however ready to be implemented and used by the requesting firm.

3.5 Collecting Data

We used four main sources for collecting data.

- A research log kept daily by the researchers focused on the themes from the hypothesis of the experiment, including observations and reflections of the ongoing development process.
- A questionnaire, that each developer answered three times per iteration (start, middle, end), 41 questionnaires in total. The questionnaire provided evidence on: A) How much Developer Stories relatively to the other practices contributed to each developers knowledge of the system, and B) Whether the systems architecture was perceived similarly by the developers, and how this perception evolved.
- The Architectural game was video taped along with the weekly meetings.
- As the development team employed a source revision control system, the source code at different revisions constituted the final data source.

The researchers were present during all work hours, taking actively part in the project, which constitutes a source of rich but informal data.

3.6 Analyzing Data

We analyzed data looking for support of the hypothesis (knowledge sharing and embodiment, and visibility of refactorings).

To provide a historical reference to the remainder of the analysis, we created a timeline using the logbook, and the artifacts produced throughout the development process (story index cards, flip charts, etc.).

We then reviewed the logbook and video recordings, mapping evidence to the main themes of the hypothesis; interactions between developers (sharing and embodiment of knowledge), interactions between developers and the customer (visibility of refactorings), and signs of architectural change (Developer Stories).

On the flip side of the questionnaire, the developers depicted their current perception of the architecture. We used these drawings to analyze how the perceptions evolved over time and how they converged or diverged between developers.

We analyzed the developers own subjective and relative rating of the contribution to their understanding of the system from Developer Stories, compared to other practices.

We even analyzed the different builds of the system with a code analysis tool (Structure101 (Software, 2006)) to see if refactorings due to developer stories had effect on the actual architecture.

The data analysis was iterative going through several of the data sources more than once in the light of the findings from the other sources.

4 FINDINGS

Presented below is the most significant findings grouped into four categories – three relating to different parts of the Developer Stories process (see Sections 2.2.1, 2.2.2 and 2.2.3), and one with findings applicable to all parts.

4.1 Acknowledging Architectural Issues

We found that knowledge of implementation challenges, design problems and refactorings was shared through osmotic communication (Cockburn, 2006), and the use of the flip chart and blackboards. Typically a discussion about these architectural issues would evolve rapidly from involving one pair to engaging one or more persons from the other pairs, and at some point two or three people would move to the stand-up area and employ either the blackboard or the flip chart.

Considering the whole span of development, the discussions were spread evenly throughout the iterations, and all developers were found to engage actively in this sharing of knowledge.

The on-site customer, being present in the room, observed the discussions about architectural issues when they took place in the stand-up area, and occasionally was able to contribute some knowledge.

There were instances of the customer instigating a discussion of architectural issues among the developers, when some functionality was demoed to the customer, and the customer asked for a change in e.g. user interface design, functionality or otherwise.

4.2 The Architectural Game

The developers all engaged actively in discussions about architectural issues experienced in the previous iteration.

The flip charts and other artifacts that were created during the iteration leading up to a particular architectural game were used by the developers as reminders of the experienced architectural issues.

The first architectural game was guided by the coach, and we observed how the developers quickly and actively engaged in the process, effectively acting as a self-organizing team.

Video recordings from the architectural games showed a noticeably quick creation of a common and high level of understanding of the current system during the review of the flip chart. The developers obtained this understanding by actively seeking knowledge from each other about aspects of the system that were currently unclear to them.

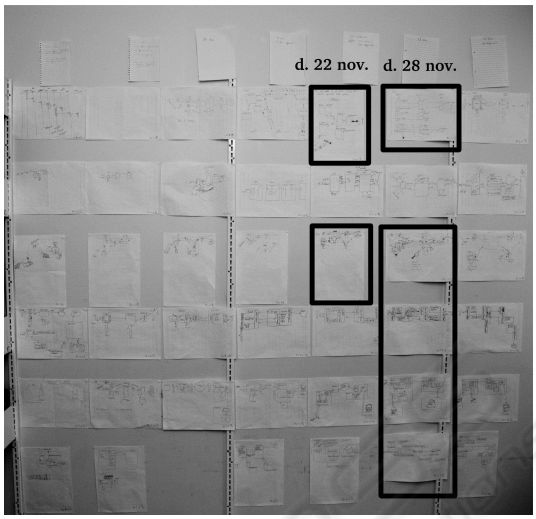

Figure 4: Drawings of the system architecture hung on a wall. Each row corresponds to a developer, time increasing from left to right.

Frequently we also observed developers feed the discussion with personal knowledge about i.a. a design pattern that had not been previously considered.

We observed the developers changing rapidly between different levels of abstraction – ranging from overall architectural patterns to implementation details.

We observed the customer choosing between the presented developer stories. When he was not immediately able to assess the business value, he challenged the developers, who where able to present the developer stories in terms that the customer could translate into business value.

4.3 Implementing Developer Stories

We found the knowledge of an architectural change happening due to a developer story spread very quickly among the developers. The black markings in Figure 4 shows how the knowledge of a particular architectural pattern described in a developer story spreads among the developers. On the 22^{nd} of November two developers had acknowledged and depicted the architectural change. The 28^{th} of November (one working day later), five developers had acknowledged the change. While the developers rotated in pairs frequently, not all the developers that acknowledged the architectural change had actually worked on that particular piece of code.

We found several situations where the customer was involved in the implementation of developer stories, just like user stories, either through discussions or direct questions concerning uncertain matters.

We analyzed a particular subset of the developed code in two different versions, and found a significant reduction of coupling and an enhancement of cohesion as an effect of a major refactoring in the third iteration. We see this as a sign of improvement of the architecture, according to the definition of architecture by the traditional quality criteria (Jensen et al., 2006). The cause of the refactoring was clearly traceable to the implementation of a specific developer story.

4.4 Generally Applicable Findings

From the questionnaires, we found that the developers themselves subjectively rated the Developer Stories practice relatively higher than other XP-practices, as a mean to increase their personal knowledge of the system, and more so more and more over the iterations.

5 RESULTS

Reflecting upon our findings, the following section summarizes the results of the experiment with regards to the hypothesis: “That Developer Stories contribute to the development process with knowledge sharing and embodiment of architectural issues among the developers, and gives the customer visibility of the possible architectural changes”.

5.1 Knowledge Sharing and Embodiment

There is widespread evidence that the practice Developer Stories did heighten the knowledge sharing and embodiment throughout the whole project by providing occasion and tools for supporting and informally documenting the discussions. Even the developers themselves considered Developer Stories as a practice that contributed to their knowledge of the system, to a larger extent than other XP practices.

The knowledge sharing and embodiment processes was initiated during the iteration leading up to the architectural games. The process occurring during the iteration was characterized by being problem oriented, and not necessarily involving all developers. When the process continued in the architectural game, focus gradually shifted to become more solution oriented, and being communal, involving all developers. The rapid shifts in levels of abstraction throughout the discussion indicates that sharing and embodiment of knowledge was actually achieved (Guindon, 1990).

When changing the architecture by implementing a developer story, the developers acknowledged the new architecture faster than otherwise expected. This indicates that the developers effectively embodied knowledge about architectural development that occurred due to developer stories.

The Architectural Game resulted in a shared and deep knowledge of the constituent parts of the system. Moreover, the developers perception of the architecture was uniform. Keeping the definition of architecture as a common understanding of the most important elements in mind, this means that the developers where in fact able to communicate about the same architecture, enabling them to make a uniform, disciplined and deliberate change of the architecture.

5.2 Visibility of Refactorings

The customer participated, through observation, instigation, control and sparring in all three parts of the Developer Stories process. This working pattern gave the customer an influence on both choice and implementation of developer stories equal to that of user stories, and some insight into the creation process.

The participation provided the customer with the insight needed for choosing what to include and to exclude (e.g. to avoid gold-plating) in the different iterations. This way the visibility combined with the customers knowledge of the systems context enhanced the architecture of the system, as decisions can be fully informed.

The act of choosing which developer stories to implement gave the customer a very tangible lever with which to control the direction of development in architectural matters.

As a positive side effect, the developers gained knowledge of the rationale on which the choice of developer stories was made, rendering them more empathetic towards the choices.

5.3 Effect of Developer Stories

When implementing developer stories, the architecture was evidently improved, but while we do conclude that the improvement in architectural quality we found, was due to implementing a developer story, we are not able to state for certain that the same refactoring had not happened in some other course of development.

However, based on this finding and on our general belief that communication raises the possibility of informed and thus wiser decisions, we do speculate that implementing the Developer Stories practice is likely to heighten quality of the architecture early in a development process by providing occasions for disciplined elaboration of the architecture.

6 DISCUSSION

Our results from the experiment showed that the suggested practice Developer Stories had effect on how developers worked with architectural issues of the system and also on the architecture of the system itself.

We found that developer stories helped developer and customer to focus on needed non-functional or other architectural requirements in a consistent (reoccurring once every iteration), disciplined and structured (in terms of developer stories) manner balanced with the rest of the practices in XP. Based on this we argue that Developer Stories can strengthen XP, effectively accommodating the critique of the aggressive refactoring (Cockburn, 2006) and providing a more disciplined approach to design activities as suggested (Fowler, 2004) while not jeopardizing the symbioses of the existing practices of XP.

From the suggested practice follows a high degree of visibility of needed refactoring tasks, of their value for the customer, of their cost (estimated) and of the customers eventual choice of stories. This visibility secures highly informed decisions on functionality and architecture and is thus likely to add greatly to the overall value of the system. Being continuous it could play some role of bounded requirements management, since all stories are known to the decision maker and he in turn is actively consulting and consulted by the developers.

We also found that Developer Stories affected knowledge sharing amongst the developer group and with the customer as well, due to his presence through the Architectural Game. Building and maintaining a uniform perception of the architecture is a prerequisite for making meaningful and consistent architectural changes. We also think this practice through supporting building of shared abstract common understanding of the system can add what was lost when the Metaphor practice was excluded from XP.

We can however not conclude anything regarding the actual quality of the architecture according to the classic criteria (Jensen et al., 2006). Securing quality is not guaranteed by following the practice as such, but the practice makes it possible and maybe even more likely.

A consideration regarding the experiment setup is, that the on-site customer was technically knowledgeable. As such, we do acknowledge that communicating developer stories in terms that the customer may translate to business value is a major challenge for the involved parties. Considering the course of events during the experiment, we do however believe that this communication can be mastered by both developers and customer after a period of practice.

It should be recognized that a control group might have been employed to compare the performance of Developer Stories. Such a control group was not however within the scope of the research project. We do still consider the results of the experiment valid, due to the fact that we do only conclude upon the presence of knowledge sharing and embodiment, and visibility, not upon general positive or negative affects on the architecture when using Developer Stories.

7 CONCLUSION

We have experimented with integrating the practice of Developer Stories into XP to investigate the effect. The experiment was a full-scale XP development project of 3 iterations, 6 developers, one on-site customer and a coach, working on at real industry request. The product is now being implemented in the requesting firm. From the experiment we found that the practice Developer Stories contributes to the development process by: 1) Heightening visibility of refactorings for both customers and Developers, 2) facilitating sharing and embodying of knowledge about architectural issues and 3) creating occasions for a disciplined elaboration of the architecture. We speculate that by this contribution, Developer Stories give development teams better means for achieving viable architectures in developed systems.

8 FUTURE WORK

Investigating how Developer Stories may be used in agile development methodologies other than XP is an open question, and subject to ongoing work.

Gaining further insight into the effect of Developer Stories may be carried out by using Developer Stories in real-world projects. Experiences from any such activity would be very interesting, and may well be reported communally on `www.developerstories.org`.

REFERENCES

Beck, K. (1999). Embracing change with extreme programming. *Computer*, 32(10):70–77.

Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

Beck, K. (2004). *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2 edition.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development. http://agilemanifesto.org.

Cockburn, A. (2006). *Agile Software Development, The Cooperative Game*. Pearson Education, Inc.

Eberlein, A. and do Prato Leite, J. C. S. (2002). Agile requirements definition: A view from requirements engineering. In *TCRE'02*. http://www.enel.ucalgary.ca/tcre02/.

Fowler, M. (2001). Design: Reducing coupling. *IEEE Software*, 18(4):102–104.

Fowler, M. (2003). Who needs an architect? *IEEE Software*, 20(5):11–13.

Fowler, M. (2004). Is design dead. http://martinfowler.com/articles/designDead.html.

Galliers, R. D. and Land, F. F. (1987). Viewpoint - choosing appropriate information systems research methodologies. *Communications of the ACM*, 30(11):900–902.

Guindon, R. (1990). Designing the design process: Exploiting opportunistic thoughts. *Human-Computer Interaction*, 5:305–344.

Jensen, R. N., Møller, T., Sönder, P., and Tjørnehøj, G. (2006). Architecture and Design in eXtreme Programming; introducing "Developer Stories". In *Lecture Notes in Computer Science : Extreme Programming and Agile Processes in Software Engineering*, pages 133–142. Springer Berlin / Heidelberg.

Lippert, M. (2004). Towards a proper integration of large refactorings in agile software development. In *Lecture Notes in Computer Science*, volume 3092, pages 113 – 122. Springer-Verlag.

Lippert, M., Axel, Schmolitzky, and Züllighoven, H. (2003). Metaphor design spaces. volume 2675 of *Lecture Notes In Computer Science*, pages 33–40. Springer-Verlag.

Paetsch, F., Eberlein, A., and Maurer, F. (2003). Requirements engineering and agile software development. In *Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, page 308.

Software, H. (accessed 29. december 2006). Structure101. http://www.headwaysoftware.com/products/structure101/.

West, D. D. and Solano, M. (2005). Metaphors be with you! In *Agile2005*.