Agile Hour: Teaching XP Skills to Students and IT Professionals

Daniel Lübke and Kurt Schneider

University Hannover
{daniel.luebke, kurt.schneider}@inf.uni-hannover.de

Abstract. Agile Methods, like Extreme Programming, have increasingly become a viable alternative for conducting software projects, especially for projects with a very short time-to-market or uncertain customer-requirements. Using a technique called Agile Hours it is possible to convey many feelings associated with an Extreme Programming project. Within 70 minutes, a project is performed in which a product is built with Lego bricks. We applied this approach to (1) students and (2) IT professionals. By comparing the two groups, we found that both behaved comparable: we observed a number of interesting differences, although of minor importance. Both groups seemed to benefit from the Agile (Lego) Hours.

1 Introduction

In opposition to the established process-oriented software-methodologies, agile methods have increasingly become common for conducting software projects during the last years. These methods are built around very light-weight techniques [1] aimed at producing high-quality software in projects with very short time-to-market, with uncertain or even unknown customer requirements [2].

All agile methods share common values which are described in the Agile Manifesto [3]. The Agile Manifesto's authors "have come to value…

- …Individuals and interactions over processes and tools,
- …Working software over comprehensive documentation,
- …Customer collaboration over contract negotiation,
- …Responding to change over following a plan."

These values have proven to be useful in scenarios where requirements are rapidly changing and huge efforts to plan upfront waste resources [4-7].

The best-known agile method is Extreme Programming (XP) [1, 8] originally introduced by Kent Beck. XP consists of 12 practices, which support each other. These practices are project guidelines prescribing how to deal with requirements, how to manage code etc.

We started using the Extreme Hour when we were teaching agile methods in a software engineering lecture. The Extreme Hour was originally developed by Merel [9]. It is a very short simulation of an XP project. In this simulation, no software is being developed but instead products are constructed by being drawn on paper. Later on, we modified the Extreme Hour and called the generalization “Agile Hours”. For example, we developed it further by using Lego bricks instead of drawing. We called this particular variant the “Agile Lego Hour”. In our terminology, Agile Hour is a more general term than Extreme Hour (drawing) and Agile Lego Hour (building).

We not only used the Agile Hours to introduce XP to (1) computer science students but had also the opportunity to teach XP to (2) IT professionals in an industrial Java User Group. In this paper we compare the behaviour of the two groups during the Agile Hours. On this empirical basis, we infer a number of lessons learned about the Agile Hour, and Agile *Lego* Hours in particular. We explored the options and potentials of Agile Hour variants. We present our observations from this exploration. Obviously, those observations need further confirmation through (controlled) experiments; we hope our findings will help to guide this continuing empirical work.

In the next section of this paper, typical problems of teaching XP in general are discussed. Afterwards, the Agile Hour is presented as our approach to deal with those problems. In the forth section, we discuss the differences between the Agile Hour and actual XP projects. Afterwards, we share common experiences of six Agile Hours we conducted. In the last two sections the differences between the students’ and IT professionals’ reaction to the Agile Hour and XP are analyzed.

2 Problem of Conveyance

2.1 Conveying XP Experience

XP favours and utilizes many social capabilities of the project’s participants. Therefore, it is difficult to teach XP without practicing it. Because many things need to be experienced in order to fully understand them and realize their consequences, it simply is not sufficient to know what practices XP comprises, what they are called and what to do. XP, like other agile methods, needs to be experienced! In this case, we mean by experience (1) an observation combined with (2) associated feelings and (3) derived reasoning and conclusions. Experiences are stronger than (theoretical) knowledge, as they are more memorable and the reasoning can be used in future situations. Agile methods (like XP) evolved from programmers’ *experiences* in the first place and can only be fully understood when combined with own experiences concerning their practices. Furthermore, many of the practices’ dependencies can hardly be inferred in theory. However, they become obvious when the method is applied to a problem.

Moreover, XP practices are completely different from other established process models, like the waterfall-model [10] or the V-model used by the German government [11, 12]. Therefore, it is even more challenging to introduce XP to IT professionals who have used and got used to the above-mentioned traditional processes and their underlying assumptions. They often do not believe that some software projects could be run this way. Some have even learned to *resent* this option.

Besides the “softness” of this topic, time-constraints are further complicating the teaching of agile methods. In university, a lecturer often faces about 100 students in a software engineering lecture, all of whom are supposed to understand agile methods within two or three weeks. In the best case, this requires an opportunity for about 100 students to somehow gain the necessary *experience*. In industry agile method courses, one does not have such a high number of participants, but their time is more expensive and, thus, even more limited. Employees are rarely assigned for a longer time just to see whether a new methodology is good or not. Neither a large number of students nor highly expensive IT professionals can conduct real XP projects for learning only.

2.2 From the Extreme Hour to the Agile Lego Hour

The first time we had to teach XP was in an introductory level Software Engineering lecture, which includes a chapter about Agile Methods. Because of the above-mentioned problems, we decided to organize the corresponding exercises as Extreme Hours. An Extreme Hour lasts 70 minutes. Because of this, Extreme Hours are an easy and time-effective way to convey the most important aspects. Extreme Hours have been successfully used in different variations (e.g. [13]). With our own minor variations, we called them “Agile Hours” to indicate the more general approach.

During the above-mentioned lecture we organized five Agile Hours. However, we encountered some disadvantages: Students could easily cheat with drawing because drawing a computer or some kind of controller is very easy and can solve arbitrary tasks. Furthermore, we found drawing is dissimilar from programming in important aspects:

- Programming is a more constructive task than drawing,
- With computers, operations like moving, deleting and reorganizing source code is much easier than to alter a drawn picture,
- Parts of drawings cannot be easily organized in hierarchical structures like packages, modules etc.
- Programming nowadays often uses components and frameworks for solving reoccurring tasks; programmers have to search for such solutions to their problems in libraries.

Finally, the quality assurance (QA) role was a very ungrateful role to play: The students doing the QA job did not participate at the planning game and development, and therefore could not participate in the most important activities.

To eliminate this effect, we decided to modify the Agile Hour more drastically. We replaced drawing by the use of Lego bricks and removed the QA role completely. The usage of Lego bricks addresses the outlined problems with drawings:

- Lego bricks are assembled in a constructive manner,
- Lego bricks can be easily removed or shifted around in a model,
- Lego bricks can be assembled to modules,
- Lego bricks are predefined components of which limited types are available.

We call this new variant “Agile Lego Hour”, as it uses Lego bricks. We conducted two Agile Lego Hours in different environments: One with 6 students, the other with 11 IT professionals during a Java User Group meeting. In the remainder of this paper we focus on comparing Agile Lego Hours in the two different environments.

3 Description of an Agile Lego Hour

For helping participants to understand agile methods, we introduced agile methods with a strong focus on XP before starting with the Agile Hour. The lecture provided the students with necessary theory.

For the IT Professionals we did a 30-minute introduction explaining the basics of XP and the origin of agile methods.

Afterwards, the participants chose their roles: For Agile Hours, two customers are needed; the rest of the participants work as developers. We acted as trackers and coaches who supervised the project and answered questions concerning the method. After the roles had been assigned, the project goal was given to the whole team. These project goals should be mechanical items buildable with elementary Lego bricks and should offer enough freedom for the customers to shape the project according to their ideas. We found “Mosquito Hunter” and “Family Spaceship” appropriate project goals because they represent general ideas, everyone has a general understanding of their functionality, and they call for mechanical (e.g. Lego) implementation.

The following main part of the Agile Lego Hour is divided into 7 phases of exactly ten minutes (and zero seconds!) each. In each phase the remaining time is projected onto a wall and as such is visible to all participants. The phases are:

1. **Story Cards & Spike:** In the first ten minutes, the customers write down their story cards for the given project idea. This is done on a flipchart, while reserving some space to the left, which is needed later on in the planning game (see figure 1). Story cards correspond to 1-2 lines on the flipchart each. In the meantime the developers are pairing, i.e. grouping to a team of 2 developers. Each team builds a prototype independently to get ideas for the project and to get accustomed to the Lego bricks available. During this phase the trackers supervise the story card creation process.
2. **Estimation of Priorities and Effort:** In the second phase the developers present their prototypes which are destroyed afterwards. Then, the customers have to explain the story cards to the developers and to prioritize them. Three levels are available: “A” for very important/cannot ship without, “B” for important and “C” for nice to have but not necessary. The priority is written next to the story cards. After this presentation, the developers have to estimate the needed effort in points. The Agile Hour (like XP) uses an abstract effort unit, e.g. “points”. Those points are calibrated using the prototypes (coaches simply “assign” them a number of points). All further estimations are carried out *in relation* to that number. If, for example, a prototype was assigned eight points, then a story card of four points should cause half the effort.
3. **Iteration I planning:** In the next phase the customers decide which story cards they want to have implemented next. They can “buy” story cards as long as their total points do not exceed the points achieved in the prototype phase. This rule implies that developers will be able to build the same number of points again during the next iteration (constant efficiency). Customers have to select next tasks based on this assumption. Developers may be faster or slower, but the initial guess is they will work at the same

speed. Afterwards, the developers organize themselves in new pairs and plan how to develop the chosen story cards in the next iteration.

4. **Iteration I:** While the customers add new story cards to the flipchart, the developers are implementing the chosen story cards. Pair programming with Lego bricks means that one developer of a pair may search for specific Lego bricks while the other one assembles the bricks to the pair's model. Typically, at the end the whole iteration product is assembled from the pairs' models.
5. **Product Presentation & Estimation of Priorities and Effort:** In the beginning of this phase, the developers present the so far developed product and the customers are judging if it fulfils their requirements. The judgement has to be based on the selected story cards. Features are only completed if they are visibly built – no hand waving and talking about how it might work is allowed. Any missing features or other shortcomings are added as story cards to the flipchart. The points of all successfully completed story cards are summed up and can be used to "buy" story cards for the second iteration. Afterwards, the customers present the new story cards and prioritize them as in phase 2. Likewise, the developers estimate the effort of the new story cards and eventually update the points of story cards already existing. This phase is normally the one, in which time easily runs out and the trackers need to speed up the process causing stress in the development process.
6. **Iteration II planning:** Customers choose story cards to complete in the second iteration. Again, the estimated effort (points) of these story cards must not exceed the points completed in the first iteration. Developers arrange in new pairs and plan how to implement the chosen story cards.
7. **Iteration II:** The second iteration is carried out like the first one, except the customers do not need to create new story cards. Instead, they are able to look at how the development is done and are able to get an impression of pairing.

After the second iteration the developers again present their product and the customers must decide whether to accept or to reject it. In all cases, the customers could accept the developed product although some small issues remained which would need to be fixed in a future iteration.

Finally, participants discussed their experiences and *reflected* on what had happened during the agile hour. We helped them to emphasize the parallels to XP projects. Reflection is very important, as it reaps the benefits of experiences: during reflection, the above-mentioned "reasoning and conclusions" are derived that help participants in future (real) projects. Emerging discussions answer questions and clarify misunderstandings and unclear aspects. For example, having only one stick per pair was mentioned as awkward. Students found it hard during integration of their results to be unable to "contribute" by drawing themselves. This phenomenon can be traced back to a real development situation in which there is only one computer per pair. In another example, customers complained about the short story card format they were forced to use. During reflection, they realized that real story cards are almost as short as our lines on the flipcharts, so the short format – and the restrictions associated with it – stays the same. Third, quality requirements caused problems: they were treated like normal story cards, but were highly orthogonal in nature. Reflection made obvious that this effect was again not a fault of the agile hour but a phenomenon waiting in real projects, too.Figure 1 shows a part of a prioritized story card flipchart, and the final result of an "automated mosquito hunter".

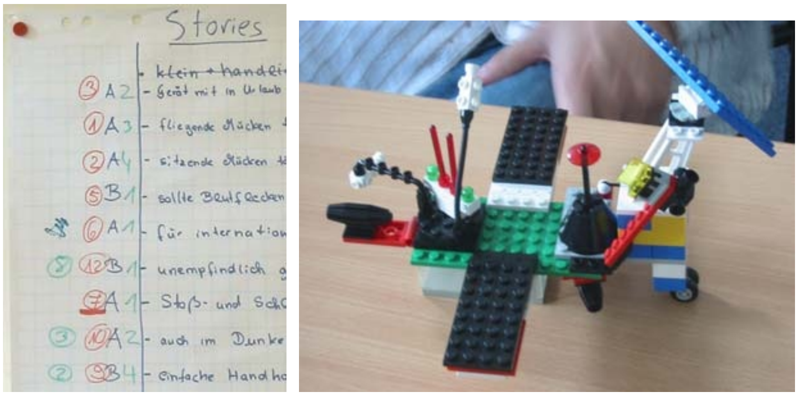

Fig. 1. (a) Story cards with priorities and estimated effort (b) Mosquito Hunter built with Lego

4 Agile Hour as a Model of XP Projects

Agile Hours (in particular Agile Lego Hours) are models of XP projects. Because of this, it is necessary to question (a) for whom this model is informative, (b) what the relevant attributes are, and (c) what model attributes do not match reality. Relevant attributes of the model (the Agile Hour) correspond to relevant attributes of reality (XP projects), and observations in the model will be mapped back to reality. Both model and reality have several attributes that are not relevant. Those attributes must not be mapped back, as there is no reasonable mapping. For example, building with Lego can be mapped back to writing Java code (relevant), but the weight of a Lego brick may have no (relevant) correspondence in an XP project.

- Agile Hours are designed to convey relevant properties of XP projects to people, who do not have any experiences in agile, especially XP, projects.
- The model focuses on the process, and not on the programming tasks. Because of this, relevant activities are e.g. *pair programming*, *onsite customer*, *story cards*, *planning game*, and *time pressure*. All of them except the last one are XP practices [1].
- Other practices are not included in the Agile Hour, like *test first*, *coding standards*, and *40 hour week*, because it is not possible to simulate them in the given time-frame or they are specific to programming.
- A summary of the Agile Lego Hour's attributes is given in table 1. Before conducting the Agile Lego Hours, we expected the *planning game*, *pair programming*, and the *onsite customer* to be the dominant attributes which the participants would recognize.

Table 1. Relevant and non-relevant attributes of the Agile (Lego) Hour

Attribute	Type	Relevant	Explanation
Acceptance Tests	XP Practice	Yes	Customers decide after each iteration if the features are working as expected or problems arise out of the implementation.
Onsite Customer	XP Practice	Yes	Customers are available for questions all the time
Pair Programming	XP Practice	Yes	Developers are grouped in pairs and have different tasks in that pair. Pairs are changing between iterations and prototype.
Planning Game	XP Practice	Yes	Efforts are estimated using a relative metric which is refined during the process. Customers pick the most business-critical stories and prioritize requirements.
Short Releases	XP Practice	Yes	Iterations only last exactly 10 minutes allowing immediate customer feedback.
Spike/ Prototype	XP Practice	Yes	A prototype is built upfront in order to explore the problem domain.
Story Cards	XP Practice	Yes	Requirements are collected story card-like at a flipchart.
Time Pressure	Project Constraint	Yes	Time pressure is always applied by the minimal time window of 10 minutes and the projection of the remaining time.
Uncertain Requirements	Project Constraint	Yes	Customers do not know the exact requirements before project start.
Continuous Builds	XP Practice	Partly	Lego Models are integrated in a short period of time but due to the time-constraint mostly at the end of the iteration like in normal projects.
Embrace Change	Agile Value	Partly	During iterations the customers are allowed to introduce new story cards contrary to existing requirements. However, only two iterations are carried out.
Refactoring	XP Practice	No	Refactoring in XP Projects is done during the test-first cycle while “refactoring” in Agile Hours is only done to integrate further components.
Test-First	XP Practice	No	Lego constructions cannot be automatically tested; therefore, test-first development is not possible.
Coding Standards	XP Practice	No	No Coding Standards can be applied for building Lego objects.
40 Hour Week	XP Practice	No	The Agile Lego Hour is too short to simulate a 40 Hour Week.

Because agile methods and XP in particular are very useful for time-limited projects and unsure customers, two relevant attributes are the *time pressure* and *uncertain requirements*.

5 Experiences with Agile Hours

While conducting Agile Hours (and by comparing ***Extreme*** Hours with Agile ***Lego*** Hours, in particular), we recognized some pitfalls in both and some decisive differences between them.

- The first problem is the **QA role** in the Extreme Hour. If there is an odd number of participants, introducing a QA role for it, looks nice at first glance. However, the role is unsatisfying from a learning perspective, since QA people do not participate in important parts, like the Planning Game. Therefore, the learning effect is worse than with participants being developers or customers. Because of this, we came to avoid the QA role in later Agile Hours and had fewer problems with this setup.
- Another set of problems comes with **pair programming**: Participants do not like to switch pairs (relevant effect: occurs in XP projects, too) or are sometimes even falling back to teamwork with 10 people building one big team. This is an irrelevant effect of the model only: In real XP projects, it is impossible to program with 10 persons at a computer, however building with Lego bricks or drawing on a sheet of paper is. In this case, the trackers have to try to encourage pairing. Furthermore, the room should be set up to hinder full-team work. For example, Lego bricks should not be put on one table but instead be left in distributed boxes for pairs.
- Concerning **story cards**, trackers should take care, that story cards really define *functional* requirements. If there are too many quality story cards, like "machine must not hurt children", it is very difficult to realize these. This is especially true for Agile Lego Hours, because it is nearly impossible to model quality properties, which is a slight disadvantage compared to Extreme Hours with mere drawings.

However, there are many **advantages** for Agile Lego Hours outweighing the disadvantages:

- For using Lego bricks we assume team-sizes may be smaller: we conducted the best Extreme Hours using drawing with 4 to 5 developer pairs, while the same level could be accomplished in an Agile Lego Hour with 3 pairs using Lego bricks.
- If one reduces special-purpose Lego bricks (e.g., looking like computers or antennas), developers cannot cheat by claiming powerful components. In Extreme Hours developers often drew computers, chips, sensors and controllers which, as they said, could literally do everything during the acceptance tests. This drives the approach to the absurd. With limited Lego bricks pretending unlimited features is less tempting.

- It is as hard to destroy prototypes as in real life (relevant effect): Whereas prototype *drawings* were put aside easily, destroying Lego prototypes in front of the participants really hurts. Like in real projects there is a big temptation to stick with prototypes despite their declared mission: being thrown away.

Finally, we found Agile Lego Hours to be more fun than Extreme Hours. While this observation may be very subjective, it was confirmed by comparing 24 questionnaires from Extreme Hours with 17 questionnaires from Agile Lego Hours. Lego bricks seem to stimulate a certain play instinct and are better received.

The essential reflection phase ties participants´ experiences from "playing with Lego" back to the serious world of software development. We consider it an advantage to provide a fun learning environment – given participants recognize its implications for their (serious) studies and work.

6 Differences Between Students and IT Professionals

While doing experiments, workshops or alike with students, often the question is raised to which extend these results are transferable to the professional area. This question is justified because both groups are different. Very different is their *background*: IT Professionals have more experience and have normally better knowledge about tools, programming languages and project management – it has been their business for several years if not decades. In contrast, students are more used to learning, are more willing to adopt new practices, and are even better at some new methods. Both groups have in common an interest for IT and related subjects.

Furthermore, IT professionals are normally older than students. Because of this, it is not necessarily sure, that playful methods being successful with students really work with IT professionals as well.

We had the opportunity to conduct an Agile Lego Hour with IT Professionals participating in an industrial Java User Group. While the Agile Lego Hour took place in the same way like the students' ones, at some points the IT Professionals behaved differently and had other problems, which are certainly related to their experiences:

- The main difference was that IT Professionals were stuck much more to an opposition of the roles of developer against customer. This contradicts XP practices in which developers are required to communicate and collaborate with the customer. However, IT Professionals wanted crystal-clear contracts in their story cards.
- Especially the story-cards and the relative effort estimation seemed to be a major annoyance. The problem with story cards was that they are too sketchy and not every detail is written down. For people who are only used to binding, detailed contracts it is difficult to get used to such a way of defining requirements ("that's not written here, so we don't have to do that", although the customer obviously meant it that way).
- Problems also arose with effort estimation in XP. It is based on relative estimation of points assigned to story cards. This makes cost- and time-estimations for comprehensive functionality upfront impossible. The ability

to estimate cost and time has been considered very important by the IT professionals.

- Team organization between students and IT Professionals was different, too. Students normally organize in groups for learning and know each other quite well. Because of this, pairing was easy and the teams were focused on their tasks. Contrary, the group of IT professionals spent most of the time discussing solutions and organizational questions which probably is related to the "Big Design Upfront"-approach normally taken. This was inherently apparent during the prototype phase, in which an IT professional pair sorted bricks according to their size to be better prepared for the project, instead of exploring the problem domain. The reason given for this behaviour was that unless "I really know what the customer wants I benefit the team most by preparation". In the second iteration, IT Professionals reverted from pair programming to team work which ended in a group meeting inefficiently discussing implementation problems.

All in all, students seemed to be more pragmatic which seems to be a beneficial attitude during Agile Hours and XP projects. Consequently, students realized more customer requirements during the two iterations, although the team compromised fewer developer pairs.

However, the IT Professional group reflected better about their Agile Lego Hour and about XP: During the reflection participants were aware that the second iteration went wrong and was inefficient. Furthermore, better questions and better realization what XP would change compared to their existing processes created a better discussion with them than with the students.

7 Analysis of the Questionnaires

After each Extreme and each Agile (Lego) Hour we handed out questionnaires asking the participants about what XP practices they think they recognized and what they feel about the Agile Hour. For comparing students to IT professionals only questionnaires of Agile ***Lego*** Hours are considered in order to have a comparable setup (see figure 2 and 3).

Most of the XP practices were rated similarly by the students and by the IT Professionals. However, there are some slight differences: The *onsite customer* and the *planning game* have been recognized by more IT Professionals than by students. We suppose this is influenced by the different backgrounds the two groups have. For IT Professionals it is not ordinary that the customer is always reachable to ask questions while the students are used to intensive support by their lecturers.

Because the second iteration of the IT Professionals was more team work than pair programming, the answers to *pair programming* and *collective ownership* seem reasonable.

Interestingly, some practices were recognized by participants, which we thought they would not. For example, *simple design* and *40 hour week* were recognized by some IT professionals and *continuous integration* has been identified by both students and IT professionals. The *40 hour week* probably can be traced back to an additional comment on the same questionnaire which states that the time pressure is very intense and thus is very exhausting.

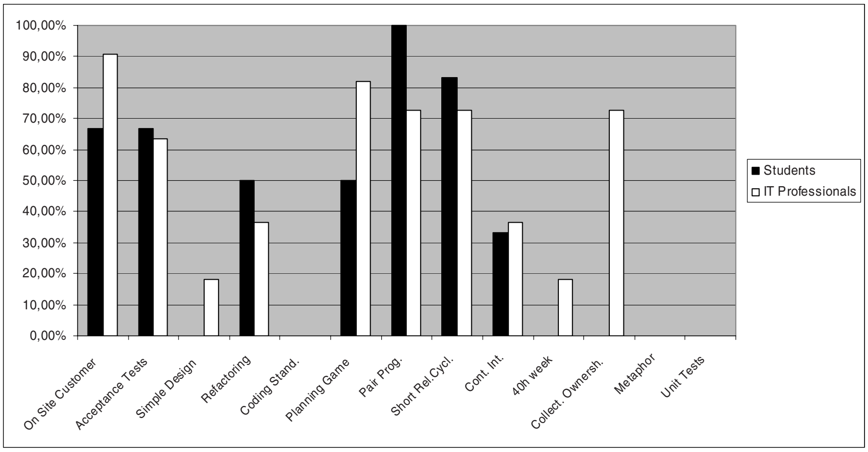

Fig. 2. Answers of the participants which XP practices they recognized

All in all, the questionnaires results confirm our expectations of the Agile Hour's most relevant attributes: Besides *short release cycles*, the *onsite customer*, *planning game*, and *acceptance tests* were recognized by most participants.

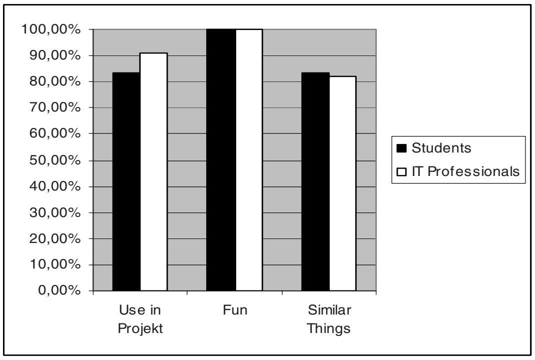

Fig. 3. Results of the Agile Hours

What both groups share is the enthusiasm about the Agile Lego Hour our as a way to learn XP: All of the participants said that they had fun attending the Agile Lego Hour and over 80% would like to attend to similar "classes" for other topics. Moreover, the Agile Lego Hour seems to be a good way to promote XP or at least agile methods: Over 80% of the students would like to use XP in some projects and even more - over 90% - of the IT Professionals answered that they would do so, too.

8 Conclusions and Outlook

In order to teach agile methods, including XP, it is necessary to not only convey textbook knowledge but also some experiences about how the methods work. This seems to be especially true for professionals using classic process models for a long time. They had a harder time to get used to the new methodologies. A very successful way of allowing these necessary experiences are Extreme Hours (drawing), Agile Lego Hours (building), or in general: Agile Hours (following the 70-minute scheme).

These *models of XP projects* allow experiencing those XP practices dealing with project management and communications. However, reflection afterwards is highly necessary in order to realize a significant learning effect. Reflection allows the participants to transfer all the experienced elements to software projects, the possible advantages and disadvantages this might have for their projects, and how this correlates with the experience they possibly have. For this, reflection does not mean filling out questionnaires but discussing and sharing the experiences and translating them back to software projects.

Encouraged by the very positive participant feedback and the interest we received, we will continue to use Agile Hours with students and with other IT professionals in the future, polishing the approach of Agile (Lego) Hours further.

Through this paper, we want to encourage others to try Agile Hours with both professionals and in student education. We hope to provide a good start by offering our insights and describing our newest variant, the Agile Lego Hour.

References

1. Beck, K., *Extreme Programming Explained: Embrace Change*. 1999: Addison-Wesley Pub Co. 224.
2. Goetz, R. *How Agile Processes Can Help in Time-Constrained Requirements Engineering*. in *TCRE 2002*. 2002. Essen, Germany.
3. Kent Beck, M.B., Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas, *Manifesto for Agile Software Development*, 2001, http://www.agilemanifesto.org/.
4. Dietmar Rohrbach, P.D.M.A. *Realisierung und Pflege von vier Versicherungs produkten*. in *Informatik 2004 - Informatik verbindet*. 2004. Ulm, Germany.
5. Martin Lippert, S.R., Henning Wolf, *Software entwickeln mit eXtreme Programming*. 2002, Heidelberg: dpunkt.verlag. 282.
6. Wright, K., *XP success story: CodeFab develops for Noggin*, 2002, http://builder.com.com/5100-6374_14-1046489-1-1.html.
7. Object Mentor, *Success Story: How Scrum + XP changed Primavera*, 2004, http://www.objectmentor.com/resources/articles/Primavera.
8. Ron Jeffries, A.A., Chet Hendrickson, Ronald E. Jeffries, *Extreme Programming Installed*. 2000: Addison-Wesley Professional. 288.
9. Merel, P., *Extreme Hour*, 2004, http://c2.com/cgi/wiki?ExtremeHour.

10. Royce, W. *Managing the Development of Large Software Systems*. in *IEEE WESCON*. 1970.
11. Versteegen, G., *Das V-Modell 97 in der Praxis*. 1999: dpunkt.verlag. 442.
12. IABG, *Das V-Modell.* 2004.
13. Silicon Valley Patterns Group, *Silicon Valley Extreme Hour*, 2005, http://c2.com/cgi/wiki?SiliconValleyExtremeHour.