Agile Modeling & EXtreme Programming

Vladyslav Pekker B. Sc.
University of Applied Sciences Constance
Brauneggerstr. 55
78462 Konstanz, Germany
agilesteel@gmail.com

ABSTRACT

Agile software development is an abstraction over the software process imposed upon rapid feedback. EXtreme Programming (XP) is a code-centric agile methodology based on the idea of using rapid feedback to its full capacity, thus applied to every single software principle, which has managed to prove itself as best practice based on the experience and commonsense of many software developers. Agile Modeling (AM) is a methodology for modeling and documentation, which can be used as an extension for almost any software process.

This paper shows how an already effective software process, such as XP becomes even more effective when combined with AM. Although XP concentrates on a certain subset of software principles, it does not comment on the remaining ones. This fact makes a lot of room for extensions, such as AM, thus making the XP & AM combination appear very natural and appealing to all stakeholders.

1. INTRODUCTION

The software industry is a very young craft. Software is a very fragile artifact. The software process is unstable and software methodologies are controversial. Software projects contain many defects, do not accomplish the tasks they were meant to accomplish and are delivered late, when delivered at all. Every stakeholder of a software project is unhappy, loses money and this clearly has to change.

Over the years a lot of effort has been made to address these problems. Scrum, XP, Dynamic Systems Development Method (DSDM), or Feature-Driven Development (FDD) and many others are steps in the right direction. These methodologies vary in many aspects, but agree on the fundamental level with each other. On February 11-13, 2001 seventeen software professionals, among them Jeff Sutherland - the inventor of Scrum and Kent Beck - the inventor of XP met to find some common ground, which would lead the software industry into the future. Naming themselves "The Agile Alliance", these people handcrafted a document, which is by now known as "The Manifesto for Agile Software Development" or simply "The Agile Manifesto". Declaring a list of principles, this document has become a beacon of hope for the software industry. Years passed. The term "agile" is polluted by the smog of certification madness, overwhelming quantity of low quality information sources, religious ignorance and simple lack of understanding the simple principles those 17 individuals put together over 10 years ago.

This paper revisits XP and augments its modeling part by explaining AM to reinforce the values of agile software development and clear the myth of extreme programmers being an old fashioned, spoiled hackers, who just do not know better.

2. SCRUM IS NOT JUST ABOUT STANDING UP AND BURNING DOWN

This section highlights the most important principles of the Agile Manifesto.

Individuals and interactions *over processes and tools.* Developing software is an intellectual and creative process, which is challenging and spectacularly complicated. Managers do not make any differentiation between a software developer and a house builder. They assume they can plug and play people into a software project as they can in the house building industry, which is obviously not the same thing. Agile teams are most commonly compared to sports teams, which can only win if the team members play together. Teamwork requires by definition a great deal of communication skills. Processes are important, but not vital. If you had a choice between an NBA level basketball team and a group of very smart, athletic and toll people who never heard of basketball before, but got it explained in detail just prior to the match, on which team would you bet your money on? And always remember, that although tools are helpful a fool with a tool is still a fool.

Working software *over comprehensive documentation.* The goal of software development is software, not documentation, otherwise it would be called documentation development.[1] Documentation is important, but not a priority. And why should it be? If you think about it, working soft-

ware is even more explanatory than a technical, gibberish document.

Customer collaboration *over contract negotiation.* Developing software is a creative process and the fact that software developers are the only stakeholders in the software project is nothing more than an illusion. The customer **develops** ideas and the software developer implements them. Thus the customer becomes a vital part of the team and communication skills are required yet again. The customer has to be creative himself. Since creativity is an iterative process, there is just no way, the customer could communicate his wishes to the software developer upfront. And even if he could, would the software developer go ahead and build the whole system in one sit?

Responding to change *over following a plan.* The fact that change **is** the state in the software world is undeniable. Technology, business environment, skills, domain understanding change over time and there is nothing you can do about it. In fact you should embrace change and act now instead of planning for tomorrow to paraphrase the surprisingly effective way of thinking: "Implement for today, design for tomorrow."[3]

3. EXTREME PROGRAMMING

This section introduces XP and explains the origins of the *extreme* part in its naming.

The main purpose of XP is to reduce project risk, making everyone happy on its way. The programmers work on tasks that really matter. These tasks are fun and fun is the reason why they got into programming in the first place. In addition to that, they never have to make important decisions on their own and therefore become fearless of change. The customers and managers receive rapid feedback, which gives them the ability to steer the project in the right direction or even entirely change the course of action.[3]

XP maximizes agile principles and practices buy boosting them to extreme levels and this is where the name comes from:

- If code reviews are good, we'll review code all the time (pair programming).
- If testing is good, everybody will test all the time (unit testing), even the customers (functional testing).
- If design is good, we'll make it part of everybody's daily business (refactoring).
- If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality (the simplest thing that could possibly work).
- If architecture is important, everybody will work defining and refining the architecture all the time (metaphor).
- If integration testing is important, then we'll integrate and test several times a day (continuous integration).
- If short iterations are good, we'll make the iterations really, really short - seconds and minutes and hours, not weeks and months and years (the Planning Game).[3]

4. AGILE MODELING

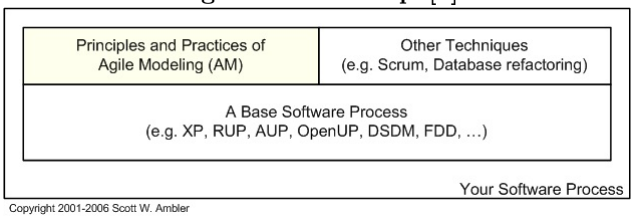
This section introduces AM and explains its goals, but before jumping into AM we discuss why bother with modeling at all.

Models are artifacts, which help you understand whay you are building or aid the communication with your team. And since communication is so important to an agile team (or any team for that matter), modeling is also important.

4.1 What is AM?

AM is not a complete software process like XP. Instead AM is a modification for a software process as presented in the following figure. AM is all about modeling and documentation. AM does not show how to create new models, but how to apply existing modeling techniques. AM

Figure 1: AM Scope[2]



- is an attitude, not a prescriptive process
- is a supplement to existing methods; not a complete methodology
- is complementary to other modeling processes
- is a way to work together effectively to meet the needs of project stakeholders
- is effective and is about being effective
- is something that works in practice; isn't an academic theory
- is not a silver bullet
- is for the average developer, but is not a replacement for competent people
- is not an attack on documentation
- is not an attack on CASE tools[1]

4.2 What makes models agile?

AM does not take the model driven approach. The audience for models are people, not computers. Therefore neither models on top of models nor a perfect specification is required. So what is an agile model? "An agile model is a model that is just barely good enough."[1] It does not even have to be graphical. The following list helps you to determine whether your model is good enough.

Agile models fulfill their purpose. Examples for purposes include: communicating the scope of your effort to a stakeholder or understanding architecture or design of a part of an application.

Agile models are understandable. The notation of your model should conform to the skills and expertise of your audience.

Agile models are sufficiently accurate. An imperfect model is better than a non-existent one. If the navigation system in your car misses a street you have a choice to either update the map or to ignore this fact if you never drive nowhere near that street anyway. But you would never throw your navigation system away because of it or would you?

Agile models are sufficiently consistent. The audience for models are people. The world is not going to come to an end if you use a dotted arrow instead of the continuous one in your UML diagram.

Agile models are sufficiently detailed. The level of detail should be chosen carefully. Remember, *just good enough* is perfect for an agile model.

Agile models provide positive value. This one is very important. If you do not know why you are creating a model or the person who requested it could not explain why he needed it, than you probably should not bother modeling. It is a business after all. Everything you do should add positive value to the project.

Agile models are as simple as possible. Many factors contribute to complexity of a model. Among them are: the level of detail or the comprehensiveness of the notation.

5. CONCLUSION

6. REFERENCES

[1] S. Ambler. *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process.* Wiley, 1 edition, 4 2002.

[2] S. W. Ambler. Am scope, 2006.

[3] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change (2nd Edition).* Addison-Wesley Professional, 2 edition, 11 2004.