How Agile Processes Can Help in Time-Constrained Requirements Engineering

Rolf Goetz
SOPHIST GROUP, Germany
rolf.goetz@sophist.de

Abstract

The requirements engineering (RE) performed when using agile processes is extremely successful in terms of efficiency and effectivity. This paper shows how and why "agile RE" helps a great deal when system manufacturers have to rush to market . We start by exploring what is the heart of the matter of RE. This will lead to a discussion of the main RE-specific aspects of agile processes. Then we will examine the conditions often found in practice in which some RE-specific agile principles cannot be fully accomplished, leading to poorer results in the field of RE. Throughout the paper we devote particular attention to approaches that offer efficiency.

The author assumes that the principles of agile software development are known to the reader.

Keywords agile processes, time-to-market, time-constrained requirements engineering, methodology, best practices, software development lifecycle

1. Exploration of Terms

Adaptive, a project or other organisation is adaptive, if it adjusts quickly to new situations and embraces change. Some authors identify adaptive with agile. Jim Highsmith introduced *adaptive* and the contrary *predicative* in [1].

Agile processes are processes like XP, Scrum, ASD, the Crystal family (see [2], [3], [4] and [5]). Although they all exist quite some time, we have never seen a 100% schoolbook-type agile project, a fact which indicates their adaptivity.

Effectivity means to "do the things right" [6]. In the Trojan War, Achilles embodies this quality. He was the greatest craftsman of the many that besieged Troja.

Efficiency means to "do the right things" [6]. Odysseus had the right idea as he designed the trojan horse. He ended the year long war.

2. RE's Core Endeavour

A great variety of methods for RE were developed and put in practice [7]. Seldom stated, the goal of requirements engineering is not to write extensive requirement documents but to effectively transfer ideas from the customer to the developer. The stronger the need is to deliver the software, the more important efficiency in transferring ideas is.

For the customer transferring ideas means to explore the universe of discourse, to build a mental reality of a future situation (the system) and to communicate that reality to the developer. The developer has to understand what is communicated, where understanding means that he has to adopt his own reality to the customer's reality as close as possible.

As no two mental models can be identical, the task is to narrow the gap between them. The various RE methods focusing on requirements representation can therefore be seen as attemps to build a stable bridge. The task under intense time-to market requirements is to build them fast.

3. RE in Agile Processes

Agile software processes are recently becoming more popular. Not being particular RE methods they propose extremely successful principles for narrowing the gap, and building bridges speeds up if they don't have to span a wide gap. This new quality is achieved because the agile software processes prefer face-to-face communication over written specifications. Face-to-face communication can be seen as a high bandwidth communication channel ([8]). Written specifications however score very low in the bandwidth category, see [5].

This section revises the peculiarities of what can be called "RE" in agile software development. We will

stress XP a bit over the other agile methodologies because it is the widest spread.

As mentioned above the most important advantage of RE in agile processes is that they rely on face-to-face communication. The main aspects are

- the on-site customer,
- frequent short releases,
- very short high level (abstract) requirement documents and
- very detailed low level (concrete) requirement documents.

The on-site customer principle suggests that at least one person from the customer's party joins the team of developers for the predominant part of the project. Preferably he sits in the same room as the rest of the team. His main task is to answer questions. He selects and sorts the requirements to implement and sets priorities by doing so. The development is therefore driven by business interests. That helps to define what is useful to the customer.

Frequently releasing pieces of code which are useful to the customer result in the ability to be faster than the competitors. One release every 1-3 months are suggested, some processes advocate one per week. Moreover, there's another positive side effect of the short delivery cycles: the requirements are better understood. Rarely the customer exactly knows the systems requirements at the beginning of the project. She is far from having a complete, concise and unambiguous model in mind that only has to be transferred to the developer. Seeing and feeling the running system helps to explore the universe of discourse and to form a mental model of the future system.

Cards with a few words on them (user stories) or use cases form the high level requirement document. These very short and abstact descriptions serve mainly as anchors or "promissory notes for future conversation" ([3]). They help communication by providing a tangible feel, as a card can be passed arround. A card, pinpointed on a wall can also be seen easily by all team members.

The customers duty in XP is writing acceptance tests, which are transformed to unit tests by the developers before any other development activity. The system has to pass all acceptance tests on every release. This ensures to always have an up-to-date specification. In our experience this quality is seldom achieved in live projects representing requirements in extensive documentation.

Acceptance test descriptions are also very suitable as a contractual base. In fact acceptance tests are nothing but a description of requirements which are at least one level more detailed than system or software requirements. They are semiformal, i.e. a quite complete, unequivocal, consistent model of the system under development.

Note that these aspects allow for a lot of knowledge to be transferred from customer to developer in a very short time with few bureaucratic overhead. Therefore we stated that the RE of the agile processes are extremely successful in terms of efficiency and effectivity.

Being very adaptive to both new general conditions and new requirements agile processes maximise the chances of a narrow gap between the customers and the developers mental model.

4. Unsuitable Situations for Agile Processes from an RE Point of View

So far this paper advocated the qualification of the agile processes concerning RE, understanding good RE as efficient communication and not producing extensive requirement documentation. However, there are situations where verbal communication alone seems to be less goal-directed. Above all two situations can be identified by looking at the RE-specific aspect of a project, omitting life-critical applications which are not built very often.

- Requirements are needed outside the team's room
- The contractual model does not allow for changing and a priori unknown requirements

The agile dogma specifically excludes these circumstances. Still the agile principles provide a considerable benefit, even if they are harder to follow. The XP answer to the need for written specification is to have a user story requiring this deliverable.

4.1. Requirements Are Needed Outside the Team's Room

Generally there is a need for extensive written specifications when the idea

- has to survive a long time (product families, traceability),

- has to travel a long way (distributed development, no on-site customer, joint international developments, especially with team members speaking different languages), or
- has more readers than would reasonably fit (or want to be) in one room.

However, there are some problems with written specifications. They should be judged against the risk of being late and the other project risks:

- Written specifications are low bandwidth communication channels! Experiment with making a compromise with video or audio channels. These media have a higher bandwidth and can also be recorded for future use.
- The longer the specification, the longer it usually takes to compile, the greater the chance of a competitor being faster. This fact is seldom recognised by organisations which insist on fixing all requirements before the development starts.

4.2. Contractual Model Does Not Allow for Changing Requirements

Some contractual models does not allow for changing requirements. The requirements must be "complete" before a contract can be made, which is often found in fixed-priced projects. The pitfall is the assumption that the development starts only after the requirements has been fixed. In reality it often has to start way before, and the requirements will almost certainly change during the other development activities.

Some useful arguments to convince the customer of the advantages of an agile process with frequent releases and frequent payments:

- pay by release: The risk is considerably lower for both parties as the customer has frequent possibilities to assess results, and always has a useable piece of system ("continuous integration"), even when he decides to quit the project. Therefore his investment is protected. The developer however does not have to make risky predictions and has more or less constant cash-flow (short releases).
- get what you want: While the customer is responsible for the acceptance of the system at all times, i.e. he cannot say that something unspecified was developed, the developer is freed from the burden to guess requirements (he cannot say the requirements were ambiguous) and can therefore concentrate on delivering a system that corresponds to the requirements by 100%. And that is exactly what the customers want since software is commissioned.
- know your rights: Maybe a lecture of the customer bill of rights [9] can help.

5. Possible Improvements

There are some additional methods to support the agile practices in the situations mentioned above. They can be used to "be" adaptive, i.e. to adapt an existing process when a project risk like a late release has to be handled. See [10] for details.

Use linguistic methods for requirements elicitation, derived from Neuro-Linguistic Programming (NLP). The customer as information source "uses" deletions, generalisations and distortions when constructing utterances from his or her underlying reality. These transformations are normal, they are needed in everyday conversation to reduce complexity (See the appendix for examples.). These transformations have a strong effect on the picture the listener or reader paints of the senders reality. In order to mutually adapt the mental models the realities of both sender and receiver should match as close as possible. As the transformations are rule-based and can be spotted instantly with some practice, the developer should learn to identify them and to scrutinise the transformed information. These methods show their strength in verbal communication, on user stories, use cases, goals, metaphors and on acceptance tests written in natural language. The result is an enhanced mental or written specification with higher communication bandwidth. Therefore these methods are particularly useful if "on-site customer" is hard to put in practice.

Clarify the non-functional requirements early in the project. An important property of non-functional requirements, especially requirements concernig the quality of service, is their great impact on the system design. As in agile processes architectures evolve over time by lots of refactoring steps, finding out about a requirement that changes the plan completely can be time consuming and costly. To avoid to be hit by a refactoring surge in later releases the developer should ask for incomplete verbs and adjectives behind which most non-functional requirements hide.

Use a template-oriented way of noting stories/use cases/tests: at least with sections for conditions, quality of service and the verb describing the actual process of

a requirement. You will write the mentioned artefacts quicker if you just fill in templates more or less.

Use integration models like object/class models or state models as a different point of view while eliciting or negotiating requirements. Under certain conditions they are appropriate to efficiently discuss things with the customer on the whiteboard: if a customer is willing to be with a bunch of developers for quite some time she may also be able to read UML diagrams.

Install exactly one stakeholder's representative. Although a single person from the customer's party may not be sufficient to cover the whole universe of discourse, you should have exactly one person in charge. This helps to avoid lengthy discussions if the person is willing and able to make decisions. Convince the customer that this person must be really bright, too.

6. Conclusion

The agile processes' principles provide a very efficient RE. Given the situation often found in practice their face-to-face aspect of communication seem to be far more suitable than written specifications. Some simple extensions of the processes, merely being techniques, can help improve performance under adverse conditions. We should also consider to replace lengthy specifications by verbal communication if time is a threat to the project.

7. References

[1] Jim Highsmith, *Adaptive Software Development*, Dorset House, New York, 2000

[2] Kent Beck, *Extreme Programming Explained: Embracing Change*, Addison-Wesley, Boston, 2000

[3] http://www.controlchaos.com

[4] http://www.adaptiveSD.com

[5] Alistair Cockburn, *Agile Software Development*, Addison-Wesley, Boston, 2002

[6] Peter F. Drucker, *Die Kunst des Managements*, Econ, München, 2000

[7] Axel van Lamsweede, *Requirements Engineering in the Year 00: A Research Perspective*, ICSE 2000, http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/l/Lamsweerde:Axel_van.html, 2002

[8] Brian Dollery, *Agile Processes*, http://www.chaosengineers.co.nz/pages/Papers/papers.html, 2002

[9] Robert Martin, *RUP vs. XP*, http://www.objectmentor.com/publications/RUPvsXP.pdf, 2002

[10] Chris Rupp, SOPHIST GROUP, *Requirements Engineering and –Management*, Hanser, München, 2001

[11] R. Bandler, J. Grinder, *The Stucture of Magic*, SBB, 1975

Appendix

This section describes examples for the three NLP-transformations mentioned in chapter 5. They are ways to reduce complexity in everyday conversation. See [11] for a much more comprehensive discussion.

Deletion. Some information is deleted from the sentence. The deleted information is either implicit knowledge or is stated elsewhere.

"The system shall display the data."

Where? To whom? When? Under which conditions?

Generalisation. A specific situation is generalised to an apparently universal "truth". The generalisation can be true ("all cars are vehicles") or wrong ("all cars have four wheels").

Distortion. (Most times in the form of a nominalisation). A process of the real world is transformed to an event. Examples are "to reserve -> the reservation", "to re-start -> the re-start" and "to return (an item) -> the return".