SOPHIST

Applying Agile Requirements Engineering to Conditions Unsuitable for Agile Development

Author: Rolf Götz
Keywords: agile processes, agile software development
Total number of pages: 11
Total number of words: 4402

SOPHIST GmbH
General Manager: Christine Rupp,
Dipl. Information Technology (FH)
Roland Ehrlinger
Vordere Cramergasse 13
90478 Nuremberg
Germany

fon: **+49 (0)9 11 40 900-0**
fax: **+49 (0)9 11 40 900-99**

e-mail: **heureka@sophist.de**
Internet: **www.sophist.de**

Copyright © 2009 by SOPHIST GmbH
This publication is protected by copyright. All rights, also transferring to translation, reprint, as well as the reproduction of certain parts of the publication are reserved. It is prohibited to reproduce, reprint or spread any part of this publication neither by using electronic systems nor by employing any other method. This is valid for teaching purposes as well. It is necessary to obtain a permission in writing! The rights of third parties are not to be affected.

V.01-0609-014

Abstract

The requirements engineering (RE) performed when using agile processes is extremely successful in terms of efficiency. This article shows how and why ‘agile RE’ helps a great deal especially 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 which are often found in practice where some RE-specific agile practices cannot be fully accomplished, leading to poorer results concerning the question ‘WHAT shall we code?’.
Throughout the article we recommend approaches that offer efficiency. The most efficient one, the use of methods for analysing natural language that are derived from psycho-therapy, are explained in greater detail.
The author assumes that the basic principles of agile software development are known to the reader.

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

0. Introduction

Agile methods for software development are widely used these days, even if only few projects use a particular agile process exactly as described in the respective books (see [1], [2], [3], [4] and [5]). In our industry projects we are often faced with two reasons for that:

- the requirements are needed outside the team’s room and
- the contract more or less relies on fixed requirements at the start of the project.

Of course the agile dogma explicitly excludes these scenarios, but still they are problems to face in practice. The polls in the Yahoo! Group on eXtreme Programming [18] seem to strengthen that finding, suggesting that only a fraction of the agile projects are 100% agile.
From a requirements engineering (RE) point of view, the RE performed in agile processes is extremely successful in terms of efficiency. This is shown by section 3 of this paper, after section 2 has stated what the problem really is that RE is dealing with. In Section 4 we explore the adverse situations mentioned above, section 5 describes some techniques which successfully helped to overcome these difficulties. One of these approaches, the method of analysing natural language by using psycho-therapy derived techniques, are extremely successful and not well known to the international RE community, they take more room in this article than the other improvements.

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. Although they all exist quite some time, we have never seen a 100% schoolbook-type agile project, a fact which indicates their adaptivity.

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.

V.01-0609-014

Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.

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. 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 due to different life experiences of every pair of persons, 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 the bridges 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 show a rather low bandwidth, see [5].

3.1 The Main Aspects

This section reviews 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[1].

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 ([6])

- the on-site customer,
- frequent small releases,
- the release planning with user stories as very short high level (abstract) requirement documents and
- acceptance tests as very detailed low level (concrete) requirement documents.

The **on-site customer** practice 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. During the release planning he selects and sorts the requirements to implement. By doing so he sets priorities. 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 ‘simply’ 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.

[1] As Ben Kovitz stated on TCRE’02, this is because XP has the best marketing...

V.01-0609-014

Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.

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

One of the customer’s duties in XP is writing **detailed requirement documents.** They take the form of 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 that represent requirements within extensive documents.

Acceptance tests 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. they form a quite complete, unequivocal, consistent model of the system under development. Unfortunately, the tests are one of the agile practices which are often omitted. Writing tests count as a painful step[2] in the agile methodology. Also the lacking tool support for automated acceptance test seems to be a problem. Note that you have to have the acceptance tests, not only if you need a contractual base.

3.2 Conclusion

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

Being very adaptive to both new general conditions and new requirements agile processes maximise the chances of a narrow gap between the customer’s and the developer’s mental models.

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

So far this article advocated the qualification of the agile processes concerning RE, understanding good RE as effective communication and not as producing extensive requirement documentation. However, there are situations where verbal communication alone seems to be less goal-directed. Above all, by looking at the RE-specific aspects of various projects not being pure agile, we identified two of these situations, 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 explicitly excludes these circumstances. Still the agile practices provide a considerable benefit, even if they are harder to follow.

[2] Daniel M. Berry advocates in [12], that every software development methodology shown up since the 1960’s has at least one painful step, a step that is avoided. This step helps to keep the amount of lead in each of the so called silver bullets ([11]) constant.

V.01-0609-014

Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.

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! Make a compromise by experimenting with video or audio channels [14]. 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 recognized by organisations which – strangely enough – 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 cashflow (*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 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, merely being techniques, 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 has to be handled, like the loss of the knowledge of the initial requirements. And they also help to make the efforts even more efficient. See [10] for details.

There are two simple things that proved to be very helpful, followed by three main improvements that stress face-to-face-communication.

- Clarify non-functional requirements early
- Use a template oriented way of noting stories
- Install one stakeholder representative
- Use integration models

V.01-0609-014

Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.

> Use methods for analysing natural language

The last suggestion will be explained in greater detail in section 5.5, because it is extremely efficient and not very well known.

5.1 Clarify Non-Functional Requirements Early

Clarify the non-functional requirements early in the project. An important property of non-functional requirements, especially requirements concerning the quality of service, is their great impact on the system design [15]. 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. Following Dan Berry, it also is painful ([12]). 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. All the ‘-ilities’ are worth a closer look as well.

5.2 Use Templates

Use a template-oriented way of noting stories, use cases, scenarios and tests: at least with sections for conditions, quality of service and the verb describing the actual process of a requirement [16]. You will write the mentioned artefacts quicker and with improved quality if you just fill in templates more or less. Furthermore, and more interesting, standards for phrasing requirements will help. They depend on the type of activity the system shall carry out:

Type of activity	**Phrasing template**
Autonomous system activity	The system shall <process?>
User interaction	The system shall provide <whom?> the ability to <process?>
Interface requirement	The system shall be able to <process?>, as soon as <condition?>

Figure 1: Phrasing Templates

Using these simple phrases ensures among other things that the actors of a function are clear. The templates can be easily enriched with conditional phrases, non-functional aspects and so on. See [10] for more details on various requirements phrasing templates and a complete rationale for using them.

5.3 One Stakeholder Representative

Install exactly one stakeholder 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. In contrast to XP’s original practice of a customer who is on-site all the time it is more realistic to ensure the stakeholder’s representative is available when needed (see [6]). This is a major improvement especially in distributed projects. It also helps to have the expert as a representative, not a trainee who was sent because it is easier for the customer to do without him.

By the way, the stakeholder’s representative can use the natural-language analysis methods described below to effectively collect the other stakeholder’s opinions.

V.01-0609-014

Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.

5.4 Use Integration Models

Use integration models like object or class models or state models as a different point of view while eliciting or negotiating requirements during the release planning [17]. From our experience 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 he may also be able to read UML diagrams.

5.5 Methods for Analysing Natural Language

Use natural-language methods for requirements elicitation, derived from Neuro-Linguistic Programming[3] (NLP) for instance [18]. We adopted these methods 1997 and fine-tuned them in the mean time. As these methods are extremely successful and not well known to the international RE community, they take more room in this article than the other improvements. You may only read the following introduction and then jump directly to section 6 to leave the details for now.

Here is what it is. The customer as information source ‘uses’ deletions, generalizations and distortions when constructing utterances from his or her underlying reality (see Figure 2).

Deletion	**Generalization**	**Distortion**
Presuppositions		
Incomplete comparatives and superlatives		
Model operators of possibility	Universal quantifiers	
Modal operators of necessity	Incompletely specified conditions	
Incompletely defined process words	Nouns without referential indices	Nominalization

Figure 2. Linguistic defects

These transformations are normal phenomena, they are needed in everyday conversation to reduce complexity. However, 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. So this attempt can be seen as a specific technique for providing questions to requirements, a step Potts and Anton [13] described as part of the Inquiry Cycle model[4] of Requirements Analysis.

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 the *on-site customer* is hard to put in practice. They also assist in clarifying the non-functional requirements (see below) and improve the overall specification skills of customer and developer.

In the next three subsections, a few of these defects and their case by case removal will be clarified on the basis of suitable examples.

[3] Neuro-Linguistic Programming is a technique used by psycho-therapists to ask what the client really means when he says something in a goal-directed manner. Vicious tongues allege the difference between a therapist and an analyst using the NLP techniques is the couch alone.
[4] The Inquiry Cycle model describes with due scientific diligence that requirements can be improved by asking questions on them and by completing and supplementing the requirements with the answers.

V.01-0609-014

Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.

5.5.1. Deletions

Deletion is a process through which we turn our attention towards selectively determined dimensions of experience and exclude others for simplification. In other words, the process of deletion reduces the world to smaller pieces, which can be handled more easily. This reduction can in certain contexts be quite helpful, but in the area of requirements for a software system, we must know exactly what information has been omitted.

The following examples exhibit various types of deletions and give an insight into the spectrum of possible representations from deletion transformations.

Presuppositions. Presuppositions are statements that must be true so that another statement has any meaning. This form of deletion is frequently found in both spoken and written requirements. Implicit assumptions are also presuppositions. For example, so that the sentence

When the altitude is less than the minimum height, an alarm should be set.

has any meaning, the statement

There is a minimal height.

must be true.

Presuppositions or implicit assumptions must be made explicit in order to complete the readers reality meaningfully. Presuppositions originate frequently through an omission by the author of a requirement because the presuppositions are either so obvious to the author that he doesn't consider it noteworthy or the author is not even aware that there is an implicit assumption.

Incomplete comparatives and superlatives. In comparison statements like

This information can also be accessed from slower storage media.

you must place yourself in the question's frame of reference. Slower than what? How slow is slow? In

The corrections should be easily modifiable.

the question remains what exactly is easily?

A comparative or superlative always requires a reference point to be completely defined. Furthermore, the unit of measure (ex. Meter, Second) and the tolerance (ex. +/- 0.1 meter, +/- 0.0001 seconds) must be declared.

Incomplete process words. Process words are those words in a sentence that describe a process. They must not necessarily be verbs, as adjectives and nouns can also play this role in a sentence (see also Nominalization). In order to be complete, a process word usually requires more arguments or perhaps even an entire noun phrase to be completely declared. Consider the following:

The system should report data loss.

The process word 'report' is completely defined only if the following questions are answered: Who reports? What is reported? How or in what manner is the information reported? To where or to whom is it to be reported? When is it reported? For how long a time is it reported?

The utilization of internal system resources should be monitored.

In this statement 'utilization' is in its noun form as a process word. To be completely defined, a process

V.01-0609-014

Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.

word must first answer the following questions: Who or what exactly is utilized? How is it utilized?

Furthermore, the process word ‘monitored’ is referential. The statement is defined completely only if the following questions are clarified: Who monitors? What is monitored? How or in which manner is it monitored?

5.5.2. Generalizations

The human ability to experience through generalization is a process that is both meaningful and necessary for survival. Through generalization, we are able to transfer an experience to related contexts. It is very important to consider the related context in which the experience is transferred, particularly in regards to the information that may have been omitted in the application of the generalization.

Through the process of generalization requirements are often made which seem to apply to a large or entire part of a system. For other parts of the system, these requirements can be very false indeed, whereas a correctly defined requirement would actually apply to a smaller piece of the system in order to have accurate scope and correct meaning.

Typical for the process of generalization is the suppression and omission of both special and error cases. In the following, the most frequently seen variations of generalization are reviewed.

Universal quantifiers. Universal quantifiers are parts of statements, which are broadly applied to all incidences of occurrence. Linguistic representatives of this group are concepts like, ‘never’, ‘always’, ‘no’, ‘every’, and ‘all’. The danger with the application of universal quantifiers is that often the specified behavior does not truly apply to all referenced objects in the group or set. The universal grouping usually contains one or more special or exception cases, which the universal quantifier references, but for those cases, the specified behavior is false.

Each signal shall additionally be labeled with a time stamp.

On the basis of the keyword, ‘each’, a question is immediately evidenced. Really each and every signal? Or are there perhaps one or more cases in which the time stamp is not required? It is critical with this type of generalization to immediately define the range of applicability so that no possibilities and occurrences of applicability are left out. Special cases must also be defined in the generalization process.

Incompletely specified conditions. Incompletely specified conditions are another indicator for a possible information loss through a generalization. The usual representations, among others, are, ‘if’, ‘then’, ‘in case of,’ and ‘depending on.’

If data X is needed by interface Y, then the time needed for a response shall be less than 0.5 seconds.

The begged question is this: which time requirements exist in the case that data X is not requested by the Y interface?

Nouns without reference index. Noun arguments without (clear) reference indices are another key type of information loss. This means that a noun exists in a statement but the noun is not sufficiently specified for clarity of the sentence.

The message shall be displayed at the working position.

The nouns of this sentence, message and working position, give no hint as to what exactly they refer. Therefore at least two questions pose themselves: Which message? Which/whose working position?

V.01-0609-014
Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.

5.5.3. Distortion

The distortion transformation appears almost exclusively in form of a nominalization.

Nominalization. A nominalization occurs when a process is reformulated into an event. A nominalization may change the meaning of a statement. A nominalization may also cause important information regarding a process to be lost. Linguistically one recognizes a nominalization as a process word (verb or predicate) that has been molded into an event word (noun or argument).

Data loss *shall be recognized and reported.*

The process behind the noun data loss actually consists of:

Data is being lost.

So this sentence has the related questions: Which data is being lost? How is the data being lost? How can the loss of data be recognized?

6. Conclusion

The agile processes' practices provide a very effective RE. Given the situation often found in practice their face-to-face aspect of communication are far more suitable than written specifications. Some simple extensions of the processes, merely being techniques, can help improve efficiency under adverse conditions:
If the requirements are needed outside the team's room or the contractual model does not allow for changing requirements, you should

- Clarify non-functional requirements early
- Use a template oriented way of noting stories
- Install one stakeholder's representative
- Use integration models
- Use methods for analysing natural language

Summarizing, we should consider to replace lengthy specifications by verbal communication if time is a threat to the project – or maybe in most other situations, too.

V.01-0609-014
Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.

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] Ken Schwaber and Mike Beedle, Agile Software Development with Scrum, Prentice Hall, 2001, ISBN 0130676349
[4] ADM, *What is Scrum?*, controlchaos.com, http://www.controlchaos.com, 2003
[5] Alistair Cockburn, *Agile Software Development*, Addison-Wesley, Boston, 2002
[6] The Rules and Practices of Extreme Programming, extremeprogramming.org, http://www.extremeprogramming.org/rules.html, 2003
[7] Alan. M. Davis, *Software Requirements*, Prentice Hall 1993, ISBN 013805763X, see bibliography
[8] Brian Dollery, *Agile Processes*, Proceedings of the New Zealand PMI Conference, 2001
[9] Robert Martin, *RUP vs. XP*, objectmentor.com, 2001, http://www.objectmentor.com/publications/RUPvsXP.pdf, 2002
[10] Chris Rupp, SOPHIST GROUP, *Requirements Engineering und –Management*, Hanser, München, 2001
[11] Fred Brooks, *No Silver Bullet*, Computer 20(4), pp.10-19, April 1987
[12] Daniel M. Berry, *The Enevitable Pain of Software Development*, Proceedings of the First IEEE Workshop on Time-Constraint Requirements Engineering, 2002, Essen
[13] Colin Potts, Kenji Takahashi, Annie I. Antón, *Inquiry-Based Requirements Analysis*, Technical Report GTI-CC-94/14, College of Computing, Georgia Institute of Technology, January 1994
[14] Ohnishi, A., *Audio-Visual Software Requirements Specification*, International Workshop on Multi-media Software Engineering, Los Alamitos, California: IEEE Computer Society, 1998, pp. 26-33
[15] Derek Hatley, Peter Hruschka, Imtiaz Pirbha, Processes for System Architecture and Requirements Engineering, Dorset House, 2000, ISBN 0932633412
[16] Juergen Dallner, Chris Rupp, *Requirements Patterns*, Proceedings of the European Conference on pattern Languages of Programming, EuroPLop 2001
[17] Rolf Goetz, *An Innovative Approach for Systems Analysis Modelling*, Proceedings of the 2nd Conference on Quality Engineering in Software Technology, ASQF, 2000, see also www.sophist.de
[18] Richard Bandler, John Grinder, *The Structure of Magic*, SBB, 1975
[19] Yahoo! Discussion Group “XP”, http://groups.yahoo.com/group/extremeprogramming/, 2003
[20] Orlena Gortel, Anthony Finkelstein, *An Analysis of the Requirements Traceability Problem*, Proceedings of the First International Conference on Requirements Engineering, pp. 94-101, 1994

Profile and Address of the Author

Rolf extensively worked on the ‘SOPHIST REgelwerk’, a set of best practices for requirements engineering, one of them being the methods for analysing natural language. He is Senior Consultant for the SOPHIST GROUP based in Nuremberg, Germany and author of several publications.

Rolf is specialized in coaching RE methodologies for national and international customers. He mainly works with natural language analysis, selected aspects of analytical philosophy and various change management practices. His employer SOPHIST GROUP develops the Requirements Engineering and Management-Tool C.A.R.E.

Contact:
SOPHIST Gesellschaft für innovative
Softwareentwicklung mbH
Vordere Cramergasse 13
90478 Nuremberg, Germany

Phone: +49 (0)911/ 40 900-0
Fax: +49 (0)911/ 49 900-99
Web: http://www.sophist.de

V.01-0609-014

Copyright © 2009 SOPHIST GmbH
For further innovative ideas all around requirements engineering visit our website www.sophist.de.