Requirements Engineering and Agile Software Development

Frauke Paetsch
Fachhochschule Mannheim
frauke.paetsch@web.de

Dr. Armin Eberlein
University of Calgary
eberlein@enel.ucalgary.ca

Dr. Frank Maurer
University of Calgary
maurer@cpsc.ucalgary.ca

Abstract

This article compares traditional requirements engineering approaches and agile software development. Our paper analyzes commonalities and differences of both approaches and determines possible ways how agile software development can benefit from requirements engineering methods.

Index Terms Requirements Engineering, Agile Development

1 Introduction

Agile software development approaches have become more popular during the last few years. Several methods have been developed with the aim to be able to deliver software faster and to ensure that the software meets customer changing needs. All these approaches share some common principles: Improved customer satisfaction, adopting to changing requirements, frequently delivering working software, and close collaboration of business people and developers.

Requirements engineering, on the other hand, is a traditional software engineering process with the goal to identify, analyze, document and validate requirements for the system to be developed. Often, requirements engineering and agile approaches are seen being incompati8ble: RE is often heavily relying on documentation for knowledge sharing while agile methods are focusing on face-to-face collaboration between customers and developers to reach similar goals. The aim of this article is to find out if some requirements engineering techniques can be used within agile development and if this could result in improvements to agile approaches.

The next section briefly gives an overview on current requirements engineering approaches. Section 3 discusses agile approaches from a requirements engineering perspective. In Section 4, we evaluate how the incorporation of some requirements engineering techniques could improve agile methods. The last section summarizes our results.

2 Requirements Engineering

Requirements engineering is concerned with identifying, modeling, communicating and documenting the requirements for a system, and the contexts in which the system will be used. Requirements describe what is to be done but not how they are implemented [6]. There are many techniques available for use during the RE process to ensure that the requirements are complete, consistent and relevant. The aim of RE is to help to know what to build *before* system development starts in order to prevent costly rework. This goal is based on two major assumptions:

- The later mistakes are discovered the more expensive it will be to correct them [3].
- It is possible to determine a stable set of requirements before system design and implementation starts.

The RE process consists of five main activities [2]: Elicitation, Analysis and Negotiation, Documentation, Validation, and Management. In the following, we will briefly examine each of these and discuss techniques that were developed for them.

2.1 Requirements Elicitation

Requirements elicitation tries to discover requirements and identify system boundaries by consulting stakeholders (e.g., clients, developers, users). System boundaries define the context of the system. Understanding the application domain, business needs, system constraints, stakeholders and the problem itself is essential to gain an understanding of the system to be developed.

The most important techniques for requirements elicitation are described in the remainder of this section.

Interviews Interviewing is a method for discovering facts and opinions held by potential users and other stakeholders of the system under development. Mistakes and misunderstandings can be identified and cleared up. There are two different kinds of interviews:

- the closed interview, where the requirements engineer has a pre-defined set of questions and is looking for answers
- the open interview, without any pre-defined questions the requirements engineer and stakeholders discuss in an open-ended way what they expect from a system.

In fact, there is no distinct boundary between both kinds of interviews. You start with some questions which are discussed and lead to new questions [2]. The advantage of interviews is that they help the developer to get a rich collection of information. Their disadvantage is that this amount of qualitative data can be hard to analyze and different stakeholders may provide conflicting information.

Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)
1080-1383/03 $17.00 © 2003 IEEE

Use cases / Scenarios Use cases describe interactions between users and the system, focusing on what users need to do with the system. A use case specifies a sequence of interaction between a system and an external actor (e.g., a person, a piece of hardware, another software product), including variants and extensions, that the system can perform. Use cases represent functional requirements of the software system and can be used during the early stages in the development process. Analysts and customers should examine every use case proposed to validate it.

Scenarios are examples of interaction sessions where a single type of interaction between user and system is simulated. Scenarios should include a description of the state of the system before entering and after completion of the scenario, what activities might be simultaneous, the normal flow of events and exceptions to the events [2].

Observation and social analysis Observational methods involve an investigator viewing users as they work and taking notes on the activity that takes place. Observation may be either direct with the investigator being present during the task, or indirect, where the task is viewed by some other means (e.g. recorded video). It is useful for studying currently executed tasks and processes. Observation allows the observer to view what users actually do in context - overcoming issues with stakeholders describing idealized or oversimplified work processes.

Focus Groups Focus groups are an informal technique where a group of four to nine users from different backgrounds and with different skills discuss in a free form issues and concerns about features of a system prototype. Focus groups help to identify user needs and perceptions, what things are important to them and what they want from the system. They often bring out spontaneous reactions and ideas. Since there is often a major difference between what people say and what they do, observations should complement focus groups.

Focus groups can support the articulation of visions, design proposals and a product concept. Additionally, they help users in analyzing things that should be changed, and support the development of a 'shared meaning' of the system [1].

Brainstorming Brainstorming helps to develop creative solutions for specific problems. Brainstorming contains two phases - the generation phase, where ideas are collected, and the evaluation phase, where the collected ideas are discussed. In the generation phase, ideas shouldn't be criticized or evaluated. The ideas should be developed fast and be broad and odd. Brainstorming leads to a better problem understanding and a feeling of common ownership of the result.

Prototyping A prototype of a system is an initial version of the system which is available early in the development process. Prototypes of software systems are often used to help elicit and validate system requirements. There are two different types of prototypes: Throw-away prototypes help to understand difficult requirements. Evolutionary prototypes deliver a workable system to the customer and often become a part of the final system. Prototypes can be paper based (where a mock-up of the system is developed on paper), "Wizard of Oz" prototypes (where a person simulates the responses of the system in response to some user inputs) or automated prototypes (where a rapid development environment is used to develop an executable prototype).

2.2 Requirements Analysis

Requirements Analysis checks requirements for necessity (the need for the requirement), consistency (requirements should not be contradictory), completeness (no service or constraint is missing), and feasibility (requirements are feasible in the context of the budget and schedule available for the system development). Conflicts in requirements are resolved through prioritization negotiation with stakeholders. Disputed requirements are prioritized to identify critical requirements. Solutions to requirements problems are identified and a compromise set of requirements is agreed upon. The main techniques used for requirements analysis are JAD sessions, Prioritization, and Modeling.

Joint Application Development (JAD) is a facilitated group session (or workshop) with a structured analysis approach. During the JAD sessions developers and customers discuss desired product features. This kind of discussion can be very productive if the session leader prevents the participants from 'running out of course'. The purpose of JAD is to define a special project on various levels of details, to design a solution, and to monitor the project until it's completed. Participants include executives, project managers, users, system experts and external technical personnel [1]. JAD promotes cooperation, understanding and teamwork among the different groups of participants. As participants should come from different backgrounds, information considering various parts of the new system can be gathered and provide a base for further requirements elicitation.

Requirements Prioritization On a project with a tight schedule, limited resources, and high customer expectations it is essential to deliver the most valuable features as early as possible. Setting priorities early in the project helps to decide which features to skip under time pressure. Requirements prioritization should be done by the customer. Both customer and developer have to provide input to requirements prioritization. The customer marks features providing the greatest benefit to users with the highest priority. Developers point out the technical risks, costs, or difficulties. Based on this information, the customer might change the priority of some features. Developers also might propose to implement a feature having a higher impact on the system's architecture but with lower priority earlier [13]. Various prioritization techniques (like pair-wise comparison or the Analytic Hierarchy Process) were developed.

Modeling System models are an important bridge between the analysis and the design process. A number of methods use different modeling techniques to describe system requirements. The most popular modeling techniques are [2]: data-flow models, semantic data models and object-oriented approaches.

2.3 Requirements Documentation

The purpose of requirements documentation is to communicate requirements between stakeholders and developers. The requirements document is the baseline for evaluating subsequent products and processes (design, testing, verification and validation activities) and for change control.

A good requirements document is unambiguous, complete, correct, understandable, consistent, concise, and feasible. Depending on the customer-supplier relationship, the requirements specification can be part of the contract.

2.4 Requirements Validation

The purpose of requirements validation is to certify that the requirements are an acceptable description of the system to be implemented. Inputs for the validation process are the requirements document, organizational standards, and organizational knowledge. The outputs are a list that contains the reported problems with the requirements document and the actions necessary to cope with the reported problems. Techniques used for requirements validation are requirements reviews and requirements testing. Requirements validation usually results in sing-offs from all project stakeholders.

2.5 Requirements Management

The goal of requirements management is to capture, store, disseminate, and manage information. Requirements management includes all activities concerned with change & version control, requirements tracing, and requirements status tracking. Requirements traceability provides relationships between requirements, design, and implementation of a system in order to manage changes to a system.

3 Agile Development Methods

In comparison to traditional software processes, agile development is less document-centric and more code-oriented. This, however, is not the key point but rather a symptom of two deeper differences between both [10]:

- Agile methods are adaptive rather than predictive. With traditional methods, most of the software process is planned in detail for a large time frame. This works well if not much is changing (i.e. low requirements churn) and the application domain and software technologies are well understood by the development team. Agile methods were developed to adapt and thrive on frequent changes.
- Agile methods are people-oriented rather than process-oriented. They rely on people's expertise, competency and direct collaboration rather than on rigorous, document-centric processes to produce high-quality software.

In this section, the most common agile methods are briefly discussed.

Extreme Programming (XP) is based on values of simplicity, communication, feedback, and courage [3]. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are. XP combines old, tried and tested techniques in such a way that they reinforce each other.

XP does not explicitly talk about requirements techniques in detail but about the general software development process and what is to be done during the process. Several XP practices (or techniques used in these practices) can be compared with slightly modified RE techniques. Specifically, during the Planning Game, elicitation techniques like interviews, brainstorming and prioritization are used. XP uses story cards for elicitation. A user story is a description of a feature that provides business value to the customer. Use cases, on the other hand, are a description of interactions of the system and its users and do not mandatory have to provide business value.

Before story cards can be written, customers have to think about what they expect the system to do. This process can be seen as brainstorming. Thinking about a specific functionality leads to more ideas and to more user stories. Every story is discussed in an open-ended way before implementation. Initially, developers ask for enough details to be able to estimate the effort for implementing the story. Based on these estimates and the time available, customers prioritize stories to be addressed in the next iteration. XP emphasizes writing tests before coding. Acceptance tests are defined by the customer and are used to validate the completion of a story card. XP is based on frequent small releases. This can be compared with requirements review and with evolutionary prototyping. The difference between XP and prototyping is, that XP requires a tested, cleanly designed code while prototypes can be 'hacked together'. Customers can review and test the functionality and design of the delivered program and discuss issues they want to be changed or added in the next release.

Agile Modeling (AM) The basic idea of AM [4] is to give developers a guideline of how to build models that resolve design problems but not 'over-build' these models. Like XP, AM points out that changes are normal in software development. AM does not explicitly refer to any RE techniques but some of the practices support several RE techniques (e.g. tests and brainstorming). AM highlights the difference between informal models whose sole purpose is to support face-to-face communication and models that are preserved and maintained as part of the system documentation. The later are what is often found in RE approaches.

Scrum is a method for managing the system development process by applying ideas on flexibility, adaptability and productivity from industrial process control theory. Scrum focuses on how a team should work together to produce quality work in a changing environment [11, 7].

The main Scrum techniques are the product backlog, sprints, and daily scrums. With regard to Requirements Engineering the product backlog plays a special role in Scrum. All requirements regarded as necessary or useful for the product are listed in the product backlog. It contains a prioritized list of all features, functions, enhancements, and bugs. The product backlog can be compared with an incomplete and changing (a kind of: living) requirements document containing information needed for development. For each sprint (= 30 day

development iteration), the highest priority tasks from the backlog are moved to the sprint backlog. No changes are allowed to the sprint backlog during the sprint. I.e. there is no flexibility in the requirements to be fulfilled during a sprint but there is absolute flexibility for the customer reprioritizing the requirements for the next sprint. At the end of a Sprint a sprint review meeting is held that demonstrates the new functionality to the customer and solicits feedback.

The knowledge gathered in the sprint review meeting and the current product backlog is used in the next sprint planning meeting. The sprint review meeting can be compared to requirements review and a presentation of an evolutionary prototype to the customer.

The Crystal Methodologies The Crystal Methodologies are a family of different methodologies from which the appropriate methodologies can be chosen for each project. The different members of the family can be tailored to fit varying circumstances. The members of the Crystal family are indexed by different colors to indicate the ”heaviness”: Clear, Yellow, Orange, Red, Magenta, Blue, Violet [8]. Up to now three Crystal methodologies have been used. These are Clear, Orange, and Orange Web. The difference between Orange and Orange Web is that Orange Web does not deal with a single project. Some Crystal Clear and Orange policy standards can be compared with RE techniques [11]:

- Incremental time-boxed delivery (Prototyping, Reviews)
- Automated regression testing of functionality (Testing)
- Two user viewings per release (Review)
- Workshops for product- and methodology-tuning at the beginning and in the middle of each increment (Review)

Feature Driven Development (FDD) FDD is a short-iteration process for software development focusing on the design and building phase instead of covering the entire software development process [11]. In the first phase, the overall domain model is developed by domain experts and developers. The overall model consists of class diagrams with classes, relationships, methods, and attributes. The methods express functionality and are the base for building a feature list. A feature in FDD is a client-valued function. The features of the feature list are prioritized by the team. The feature list is reviewed by domain members [12]. FDD proposes a weekly 30-minute meeting in which the status of the features is discussed and a report about the meeting is written. Reporting can roughly be compared with requirements tracking.

Dynamic Systems Development Method (DSDM) provides a framework for rapid application development [9]. The first two phases of DSDM are the feasibility study and the business study. During these two phases the base requirements are elicited. Further requirements are elicited during the development process. DSDM does not insist on certain techniques. Thus, any RE technique can be used during the development process. In DSDM, testing is integrated throughout the lifecycle. The philosophy of DSDM is ’test as you go’ [9]. All sorts of testing (technical, functional) are carried out incrementally by the developer and the users on the team. DSDM explicitly highlights the use of JAD sessions and emphasizes prototyping [9].

Adaptive Software Development (ASD) provides an framework for the iterative development of large, complex systems. The method encourages incremental, iterative development with constant prototyping [5]. ASD highlights that a sequential waterfall approach only works in well-understood and well-defined environments. But as changes occur frequently in software development, it is important to use a change-tolerant method. The first cycles of an ASD project should be short, ensure that the customer is strongly involved and confirm the project’s viability. Each cycle ends with a customer focus group review. During the review meetings a working application is explored. The results of the meetings are documented change requests. Although ASD explicitly refers to JAD sessions it does not propose schedules for holding JAD sessions.

4 RE techniques for agile approaches

In the previous section, we gave an overview on requirements engineering techniques as well as on agile methods. Here, we now analyze potential synergies between these approaches.

Customer involvement The CHAOS report [14] showed the critical importance of customer involvement. Customer involvement was found to be the number one reason for project success, while the lack of user involvement was the main reason given for projects that ran into difficulties. A key point in all agile approaches is to have the customer ’accessible’ or ’on-site’. Thus, traditional RE and agile methods agree on the importance of stakeholder involvement.

Agile methods often assume an “ideal” customer representative: the representative can answer all developer questions correctly, she is empowered to make binding decisions and able to make the right decisions. Even if the requirements are elicited in group sessions (DSDM, Scrum) it is not guaranteed that users or customers with all necessary backgrounds are present. On the other hand, RE has a less idealized picture of stakeholder involvement. The different elicitation techniques aim to get as much knowledge as possible from all stakeholders and resolve inconsistencies. In addition, RE uses externalization and reviews to ensure that all requirements are known and conflicting requirements are in the open

Another difference between traditional approaches and agile methods is that in traditional approaches the customer is mainly involved during the early phase of the project while agile methods involve the customer throughout the whole development process.

Interviews As customer involvement is a primary goal of agile software development, the most common RE-related technique are interviews. Interviews provide direct and ’unfiltered’ access to the needed knowledge. It is known that chains of knowledge transfer lead to misunderstandings. All agile approaches emphasize that talking to the customer is the best way to get information needed for development and to avoid misunderstandings. If anything is not clear or only vaguely

defined, team members should talk to the responsible person and avoid chains of knowledge transfer. Direct interaction also helps establishing trust relationships between customers and developers.

Prioritization is found in all the agile approaches. A common practice is to implement the highest priority features first to be able to deliver the most business value. During development, the understanding of the project increases and new requirements are added. To keep priorities up-to-date, prioritization is repeated frequently during the whole development process.

JAD sessions are used in ASD to increase customer involvement. In DSDM, JAD sessions are used to gain understanding of the new system at the beginning of a project. Focusing the interaction to short JAD sessions does not bind the customer representative as much as the on-site customer in XP. The results of the sessions are usually documented and accessible if further questions arise later. The documentation requirement should be relaxed in an agile context. To have constant feedback, JAD sessions should be held frequently during the development process. JAD sessions encourage customer involvement and trust in each other. This is clearly in line with agile principles.

Modeling Although modeling is used in AM, the purpose is different when compared with RE. In AM, models are used to communicate understanding of a small part of the system under development. These models are mostly throw-away models. They are drawn on a whiteboard or paper and erased after fulfilling their purpose. Most of the models do not become part of the persistent documentation of the system. In RE, models on different levels of abstraction are used. All these models normally become part of the system documentation and need to be kept up to date. The purpose of modeling in FDD is similar to RE: The developed model represents the whole system and development is based on this model. But as the design and build phase are iterative, models are quickly translated into running code.

Documentation In agile software development, creating complete and consistent requirements documents is seen as infeasible or, at least, as not cost effective. Some agile methods include a kind of documentation or recommend the use of a requirements document (DSDM, Scrum, Crystal) but the decision on its extend is left to the development team and not described in detail. The assumption is that the documents are much smaller than in traditional approaches. The lack of documentation might cause long-term problems for agile teams. Documentation is used for sharing knowledge between people: a new team member will have many questions regarding the project. These can be answered by other team members or by reading and understanding "good" documentation. Asking other team members will slow down work because it takes some time to explain a complex project to someone. Documentation reduces knowledge loss when team members become unavailable (e.g. they moved to another company or are working on a new project). This is particular an issue when software needs to be maintained in the long run. Long term problems might be limited as agile methods often produce clean and compact code. Additionally, customers often ask the agile team to produce design documentation before the team is resolved when the system moves to a pure production environment. The scope of the documentation is often very limited and focuses on the core aspects of the system. This increases the chances that the documentation can be kept up to date when the software is changed. On the other hand, agile methods should be more productive than traditional approaches as they spend less time on producing documentation than traditional RE approaches.

Traditional approaches try to produce enough documentation to be able to answer all questions that might be asked in the future. To be able to do so, they need to (1) anticipate future questions and (2) answer them in a concise and understandable manner. Both things are difficult – this is why writing good requirements documents is so hard. Usually, to get a good coverage of future questions, traditional approaches might produce too much documentation (i.e. answering questions that will in fact never be asked and wasting money on writing the answers). As a side effect of comprehensive documentation, it becomes difficult to keep all documents up-to-date and to find the relevant information for answering a question. Some advantages or disadvantages of documentation are related to team size. With a big team, it might be better to have documentation instead of explaining the same thing many times to different people. To summarize, agile methods tend to err on the side of producing not enough documentation while traditional approaches tend to overdocument.

Validation Agile approaches use frequent review meetings and acceptance tests for requirements validation. Review meetings show that the project is in target and schedule increasing customer trust & confidence in the team or highlighting problems very early. Agile approaches use different kinds of review meetings to present the new software. Customers can use the software, experience how the system works and determine which functionality has already been implemented. If they have questions, these can be immediately answered by the developers. Customers can also discuss the implementation with developers and ask for changes in design. During the review meetings, they learn about strength and weakness of the design and the technology and in which areas there are advantages and limitations. Customers can also run acceptance test to validate if the system reacts in the expected way and, if not, clarify the issue. The questions and change requests influence future development. Depending on the method and project circumstances, the software is put into production after each review. This fast deployment changes the economics of the software project by providing a faster return on investment. Agile development is comparable to evolutionary prototyping: both are based on delivering frequently a piece of working code to enhance feedback. The difference lies in the strong emphasis on testing in agile approaches.

Management From the RE view, it should be possible to track

changes made to requirements, design, or documentation to see why any changes were made. Creating traceability results in more work and additional documentation. Additionally, it has not yet been shown that change tracking provides any economic benefit for a project. The lack of documentation could be critical if the requirements form a legal contract between two organizations. That is one reason why agile project contracts often are based on time and expenses and not on fixed price/fixed scope. Contracts based on fixed price/fixed scope create trust by defining what the customer will get, for what price and in which time. Agile approaches create trust by tightly integrating the customer into the development process. The customer sees that the development team is working on the software and believes that the team will deliver working software that will meet his needs.

Agile methods provide a good base for requirements management. All the requirements are written on index cards or maintained in a product backlog/feature list. The difference is the level of detail in which a requirement/user story/feature is described: agile practices usually omit the details postpone the expenses for gathering the details until the requirement needs to be fulfilled in the next iteration. One can call this lazy requirements elicitation.

DSDM discusses change tracking. When parts of the business study have to be repeated, the changes can be documented in a separate document. The product backlog in Scrum could be used to version requirements changes. Instead of deleting old versions, they could be kept and refer to the new requirement with a note why they were deleted/changed.

Observation and Social Analysis, Brainstorming These techniques are not explicitly mentioned in any agile software development method but can be used with any approach. Especially, observation methods can provide benefit in requirements elicitation as a highly qualified 'user' is not always the best person to talk about her work process a they sometimes forget important details as the work has become routine. These details can be discovered by observation.

Non-functional requirements In agile approaches handling of non-functional requirements is ill defined. Customers or users talking about what they want the system to do normally do not think about resources, maintainability, portability, safety or performance. Some requirements concerning user interface or safety can be elicited during the development process and still be integrated. But most non-functional requirements should be known in development because they can affect the choice of database, programming language or operating system. Agile methods need to include more explicitly the handling of non-functional requirements.

5 Conclusion

The RE process phases elicitation, analysis, and validation are present in all agile processes. The techniques used vary in the different approaches and the phases are not as clearly separated as in the RE process - they merge in some ways (the Planning Game in XP is an elicitation and analysis approach). They are also repeated in each iteration - which makes it harder to distinguish between the phases. The lazy approach to requirements engineering might have cost advantages as it postpones the effort/expenses for gathering requirement details until the last minute: just before the requirement is implemented, it needs to be understood by the developers and then they talk with customer representatives. The techniques used in the agile development processes are sometimes described vary vaguely and the actual implementation is left to the developers. This is a result of the emphasis on highly skilled people in agile methods: "good" developers will do the "right thing". Traditional approaches, on the other hand, of prescribe details of what needs to be done and so provide more guidance to do the "right" thing. Unfortunately, determining upfront what the right thing in a given project context is, is very difficult. As RE management is based on documents, is not very well represented in agile software development approaches. Documentation is a part of agile software development but its extend is not even in the ballpark as in RE. As all agile approaches include at least a minimum of documentation, it is the responsibility of the development team to ensure enough documentation is available for future maintenance. Overall, in key areas (like stakeholder involvement) agile methods and RE are pursuing similar goals. The major difference is the emphasis on the amount of documentation needed in an effective project. This partially steams from differences in core assumptions on the stability of requirements and empirical studies on the cost effectiveness of both approaches are required.

References

[1] Linda A. Macaulay: *Requirements Engineering*, Springer Verlag, 1996.

[2] Gerald Kotonya and Ian Sommerville: *Requirements Engineering*, John Wiley & Sons, 1997.

[3] Kent BeckExtreme *Programming explained*, Addison-Wesley, 1999.

[4] Scott W. Ambler: *Agile Modeling*, John Wiley & Sons., 2001.

[5] James A. Highsmith III: *Adaptive Software Development*, Dorset House Publishing, 1996.

[6] Alan M. Davis: *Software Requirements Revision Objects*, Functions, & States, Prentice Hall PTR, 1994.

[7] Ken Schwaber, Mike Beedle: *Agile Software Development with Scrum,* Prentice Hall, 2001.

[8] Alistair Cockburn : *Agile Software Development*, Addison-Wesley, 2002.

[9] Jennifer Stapleton : *DSDM - Dynamic System Development Method*, Addison-Wesley, 1995.

[10] Martin Fowler: *The new methodology*, http://www.martinfowler.com/articles/newMethodology.html (last visited: May 2003).

[11] Pekka Abrahamsson, Outi Salo, Jussi Rankainen & Juhani Warsta : *Agile software development methods - Review and analysis*, VTT Electronics, 2002.

[12] Peter Coad, Eric Lefebvre, Jeff De Luca : Java Modeling in Color with UML, Prentice Hall PTR, 1999, Chapter 6.

[13] Karl E. Wiegers : Software Requirements, Microsoft Press, 1999

[14] Standish Group : Chaos Report, http://www.standishgroup.com.