Agile Requirements Engineering: A Research Perspective

Jerzy Nawrocki, Mirosław Ochodek, Jakub Jurkiewicz, Sylwia Kopczyńska, and Bartosz Alchimowicz

Poznan University of Technology, Institute of Computing Science,
ul. Piotrowo 2, 60-965 Poznań, Poland
{Jerzy.Nawrocki,Miroslaw.Ochodek,Jakub.Jurkiewicz,Sylwia.Kopczynska,
Bartosz.Alchimowicz}@cs.put.poznan.pl

Abstract. Agile methodologies have impact not only on coding, but also on requirements engineering activities. In the paper agile requirements engineering is examined from the research point of view. It is claimed that use cases are a better tool for requirements description than user stories as they allow zooming through abstraction levels, can be reused for user manual generation, and when used properly can provide quite good effort estimates. Moreover, as it follows from recent research, parts of use cases (namely event descriptions) can be generated in an automatic way. Also the approach to non-functional requirements can be different. Our experience shows that they can be elicited very fast and can be quite stable.

Keywords: Requirements engineering, agility, use cases, non-functional requirements, effort estimation, user manual.

1 Introduction

Agile methodologies, like XP [6] and Scrum [31], have changed our way of thinking about software development and are getting more and more popular. They emphasize the importance of four factors: oral communication, orientation towards working software (main products are code and test cases), customer collaboration, and openness to changes.

Agility impacts not only design and coding but also concerns requirements engineering [7,8]. In this approach classical requirements specification based on IEEE Std. 830 [1] is replaced with user stories [6,12] and face-to-face communication. User stories can be used for effort estimation and planning. Effort estimation is based rather on personal judgement than on such methods as Function Points [3] or Use-Case Points [17]. Moreover, in the agile approach, requirements are not predefined – they emerge while software is being developed [8].

As pointed out by Colin Doyle[1], agile requirements engineering based on user stories has some advantages and disadvantages . On the one hand, it encourages

[1] C. Doyle, Agile Requirements Management – When User Stories Are Not Enough, http://www.youtube.com/watch?v=vHNr-amZDsU

V. Geffert et al. (Eds.): SOFSEM 2014, LNCS 8327, pp. 40–51, 2014.

effective communication and better adapts to change. On the other hand, it cannot be applied in a project where a request-for-tenders is used (this requires an up-front specified requirements), the customer is not always available for just-in-time requirements discussions, or the product/project is complex (many conflicting customers, lots of requirements). Another problem is human memory: the Product Owner cannot remember all the requirements and s/he is prone to forgetfulness. In such cases documented requirements would help a lot.

In this paper a research perspective concerning agile requirements engineering is discussed. We focus on the following questions:

- **Q1-UStories**: User stories have an old competitor: use cases invented by Ivar Jacobson in the 80s [15] and later elaborated upon by Alistair Cockburn et al. [2,11]. Are user stories really the best choice?
- **Q2-NFRs**: How can non-functional requirements be elicited to obtain a good balance between speed and quality?
- **Q3-Effort**: One of the key activities in release planning (e.g. performed by playing the Planning Game) is effort estimation. Can automatic effort estimation provide reliable enough estimates?
- **Q4-Manual**: Can the effort concerning auxiliary activities, like writing a user manual, be significantly reduced by using requirements specification?

Those issues are discussed in the subsequent sections of the paper.

2 Written vs. Oral Communication

Agile approaches emphasize oral communication when it comes to requirements elicitation. User stories, which are advocated by XP [6] and Scrum [30], are not supposed to provide complete requirements, they are rather "reminders to have a conversation" [12] with the customer. User Story answers three questions: *what action is possible?*, *who performs this action?* and *what is the business value of this action?* User Stories are short, they are usually expressed in one sentence. Example of a user story is presented in Figure 1A. Similar information is presented in UML Use Case Diagrams [28]. They convey information about actors and their goals (actions). Example of a UML Use Case diagram is presented in Figure 1B. Apparently, User Stories and UML Use Case Diagrams provide roughly the same amount of information about the requirements. Therefore, UML Use Case Diagrams can be used instead of user stories.

User Stories, as reminders for future conversation, are good when a customer is non-stop available for the software delivery team. This is supported by the XP *on-site customer* practice. Unfortunately, on-site customer is rarely available in real life projects, mostly due to high costs. Therefore, more information needs to be captured during the short times when customer is available. While, with user stories only so called *acceptance criteria* can be documented, use-cases allow for more precise capture of requirements. These requirements are documented in a textual main scenario - sequence of steps presenting interaction between actor

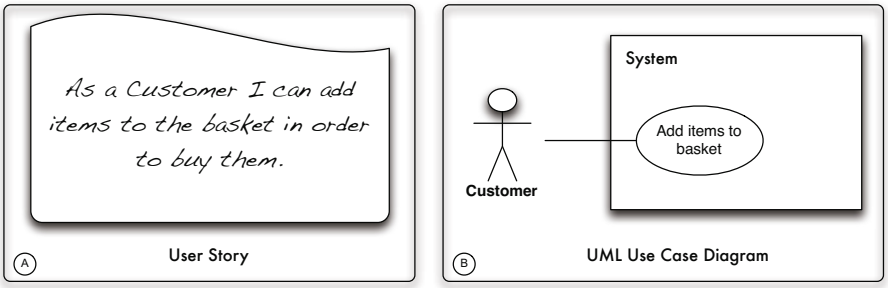

Fig. 1. A - Example of a user story; B - Example of a UML Use Case Diagram

and system. Main scenario should always demonstrate the interaction which leads to obtaining a goal by an actor.

Observation 1. Use cases are more general (flexible) than user stories as they provide an *abstraction ladder* (zooming).

The highest abstraction rung is the level of context diagram (just actors). Below is the level of use case diagrams (such a diagram can be zoomed-out to a context diagram). More details are available at the level of use case scenarios (they can be zoom-out to a context diagram), beneath which are events and alternative activities (they can be easily trunkated to main scenarios). Scenarios can be decorated with low-fidelity screen designs and that way one can generate mockups (see e.g. [22]). Those mockups can be used to elicit test cases from end users, what makes use cases testable.

2.1 HAZOP-Based Identification of Events in Use Cases

To have a complete use case one has to identify all events that can appear when use case steps are executed. A question arises: *how to identify events in use cases in effective and efficient way?*

Events in use cases resemble deviations in mission-critical systems. Therefore, a HAZOP-based method, called H4U [16], for events identification has beed proposed. HAZOP [29] is a method for hazard and deviations analysis in mission-critical systems. This method is based on sets of primary and secondary keywords which are used in brain-storming sessions in order to identify possible hazards. H4U is also built on the idea of keywords which help to identify possible events in use cases. The accuracy and speed of H4U were evaluated and compared to the *ad hoc* approach in two experiments: with 18 students and with 64 IT professionals. Based on the experiments it could be concluded that H4U method offers higher accuracy (maximum average accuracy was equal to 0.26) but lower speed (maximum average speed was equal to 1.04 steps per minute) comparing

to the *ad hoc* approach (maximum average accuracy was equal to 0.19, maximum average speed was equal 2.23 steps per minute).

2.2 Automatic Identification of Events in Use Cases

Using H4U one can achieve higher accuracy of events identification, however, higher effort is required. What if events in use cases could be identified in an automatic way? This could reduce the effort and time required to identify events. Moreover, this would mean that Analyst can focus on actors' goals and positive scenarios which require a lot of creative work and the more tedious task of events identification could be done automatically. Results from our initial research show that around 80% of events of the benchmark requirements specification [4] can be identified in an automatic way. Morover, speed of automatic events identification was at the level of 10 steps per minute. This results were achieved with a prototype tool built with a knowledge base from real-life use-cases, inference engine, NLP and NLG tools. Comparing these results to the results from the ealier mentioned experiments with the *ad hoc* and H4U method, it can be concluded that events can be effectively idetified in an automatic way. This allows Analyst to focus on the core of the requirements (use-case names and main scenarios) and have descriptions of events generated automatically.

This research poses a more general research question:

Question 1. To what extend requirements specification can be supported by a computer?

3 Elicitation of Non-functional Requirements

Although user stories are considered by XP [6] and Scrum [30] as the main tool to record requirements, as we stated in Section 3, they may not be sufficient. At first, a user story is not supposed to express a complete requirement. Secondly, it focuses on what actions and activities are performed by a user in the interaction with a system. Such approach may lead to omission of the requirements regarding *how* the functions provided by the system should be executed. According to the results of the investigation of agile projects carried out by Cao and Ramesh [8], customers often "focus on core functionality and ignore NFRs". They also found that in many organizations non-functional requirements(NFRs) are frequently ill defined and ignored in early development stages [8]. It may have severe consequences, as in the cases of Therac-25[19], Arline 5 accident[23] etc., it may lead to excessive refactorings, or cease further system development as its architecture was ill-designed based on insufficient information. However, in agile methodologies, thorough analysis and completeness of requirements (e.g. required by IEEE 830[1]) are no longer a prerequisite to software design and coding - time to market is getting more and more important. Thus, in the context of NFRs, the challenge is how to achieve *"a proper balance between the cost of specifying them and the value of reducing the acceptance risk"* [9]. There exist

a number of methods and frameworks which deal with non-functional requirements, e.g., NFR Framework [21], KAOS [32]. However, the existing methods are claimed to be too heavy-weight to be used in agile context [10], and there is little known about cost and value of using them.

To respond to the challenge and fill the gap we proposed a quick method, called SENoR (Structured Elicitation of Non-functional Requirements), dedicated to agile software development. It consists of 3 steps that are presented in Figure 2). The cornerstone of SENoR is *Workshop* which consists of: (1) a presentation of the business case of the project and of the already known functionality of the system, (2) a series of short brainstorming sessions driven by the quality subcharacteristics of ISO25010 [14] - this is the main part of the Workshop, (3) a voting regarding the importance of the elicited requirements.

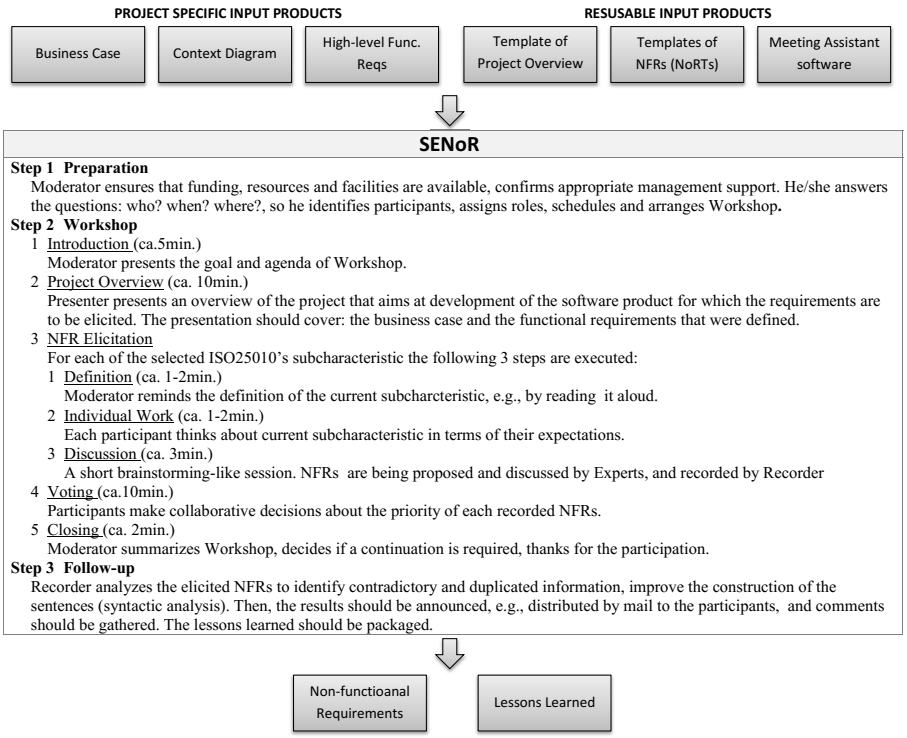

Fig. 2. SENoR – process, input and output products

SENoR workshops are supported with short definitions of quality subcharacteristics and templates of non-functional requirements. Each SENoR workshop lasts no longer than 2 hours.

The data about SENoR workshops have been collected since its first application in 2008 [18]. They have been used to improve the method. Recently,

7 agile projects run at the Poznan University of Technology have been observed. Those projects used SENoR workshops and their aim was to deliver internet applications to be used by the university for administrative purposes.

The average time of a SENoR workshop was ca. 1h 15min, (the shortest took 2854s and the longest — 7554s). 89% (34 out of 38) participants claimed that they are for organizing such workshops in their future projects. On average, 27 NFRs were defined in a workshop, and 92% participants regarded quality of the elicited NFRs good-enough (sufficiently correct and complete) to start architectural design. We also investigated stability of the elicited NFRs, i.e. how many of them 'survived' development changes. What is surprising, average stability of NFRs collected within SENoR workshops was at the level of 80%.

From the point of view of the Q2-NFRs question, the presented results are very promising:

Conjecture 1. A sequence of very short brainstorming sessions driven by quality characteristics can provide quite stable set of non-functional requirements and represents a good balance between cost (time) and quality (stability).

If the above conjecture was true, it would imply that software architecture, which strongly depends on NFRs, can also be quite stable.

4 Automatic Effort Estimation Based on Use Cases

Although project planning is not directly a part of the RE process, they both visibly relate to each other. In order to plan a project or development stage, one has to determine and analyze its scope. The connection between planning and RE is especially visible in the context of agile software development, where constant project planning is often placed next to the core RE practices [8].

Indisputably, in order to plan a project one has to first estimate the effort required to perform all the project's tasks. Depending on the chosen type of budget, an accurate effort estimation becomes more important at different stages of the project. For a fixed scope budget, obtaining an accurate estimate is already extremely important at the early stages of software development, when the crucial decisions about the overall budget are made. If an unrealistic assumption about the development cost is made, the project is in danger. Both underestimated and overestimated effort is harmful. Underestimation leads to a situation where a project's commitments cannot be fulfilled because of a shortage of time and/or funds. Overestimation can result in the rejection of a project proposal, which otherwise would be accepted and would create new opportunities for the organization. In the fixed budget approach the situation is different. The budget could be allocated in advance, but the project team is trying to incrementally understand the customer's needs and deliver as much business value as possible until the available resources are depleted.

When it comes to agile software development methods, they are naturally well suited to fixed budget projects. They assume that changes in requirements are an inherent property of software development projects, thus, it is not reasonable

to invest too much time in the preparation of comprehensive software requirements specification at the beginning of the projects, which can quickly become obsolete. As a result, project planning in agile software development is more oriented towards estimating and planning releases than the project as a whole (e.g, using the planning game, story points, planning poker [6,12,13,20]). Therefore, an important question emerges about what to do in the case of fixed scope projects being developed in an agile environment or when a customer agrees to the fixed budget approach, but would also like to know if he/she can afford to solve the problem.

An answer to the question would be to elicitate high-level requirements at the project's beginning, and then to estimate its total effort assuming the requirements forming the scope of the project. According to Cao and Ramesh [8] elicitating such requirements is not such an uncommon practice in agile projects. Still providing an accurate estimate based on such requirements is a challenge.

One of the methods that could be used for effort estimation based on high-level requirements is the TTPoints method [26,27]. It is designed to provide functional size measurement based on use cases, but contrary to other use-case-based effort estimation methods, such as Use Case Points [17], it is not strictly bounded to the syntax of use cases. Instead it relies on the semantics of interaction presented in use-case scenarios.

The main, considered unit of interaction in TTPoints is called semantic transaction. Empirical analysis of use cases has led us to define a catalogue of 12 semantic transactions-types presented in Figure 3. Each transaction type corresponds to the main intent of the interaction between the user and the system under development. This enables their identification even if the details of a use-case scenario are still to be determined. An example of a create-type transaction identified in a use case and user story is presented in Figure 4. If fully-dressed use cases are available transaction identification is more accurate and could even be automated using NLP (natural language processing) tools [24,25]. It visibly reduces the effort required to analyze a use-case-based requirements specification.

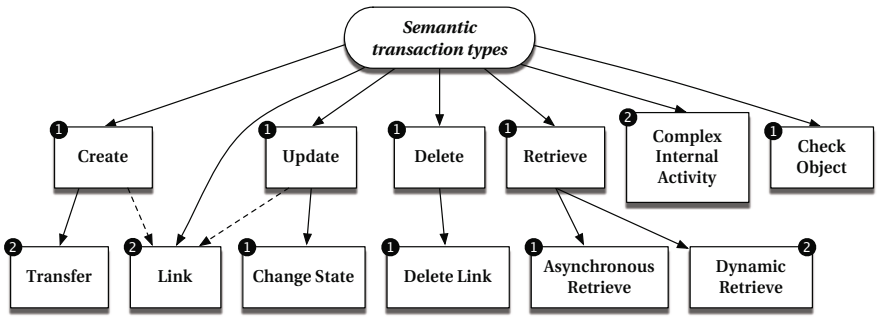

Fig. 3. Semantic transaction-types in use cases with the numbers of core actions

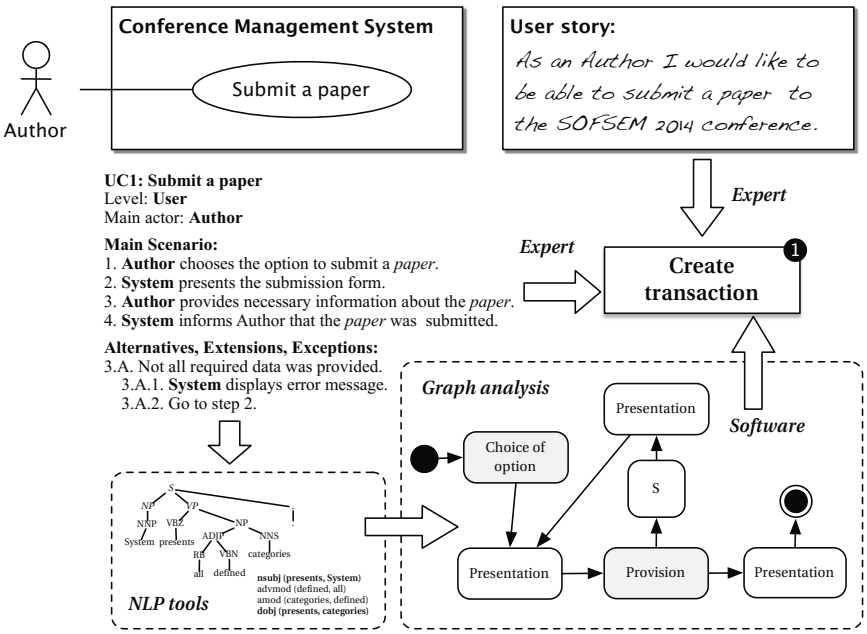

Fig. 4. An example of create semantic-transaction in a use case and user story

The next step is the analysis of each transaction. One has to determine the number of different actors that interact with the system under development and the number of domain objects being processed within a transaction.

Finally, one can calculate the functional size expressed in TTPoints according to the following formula:

$$TTPoints = \sum_{i=1}^{n} Core\_Actions_i \times Objects_i \times Actors_i \tag{1}$$

where

- n is the number of semantic transactions within the scope of the count;
- $Core\_Actions_i$ is the number of *core actions* of the i-th transaction (see Figure 3);
- $Objects_i$ is the number of meaningful (independent) domain objects processed by the i-th transaction;
- $Actors_i$ is the number of actors in the i-th transaction which cooperate with the system under development.

The TTPoints size measure can be further used to estimate development effort. If historical data is available in organization one can construct a regression model or simply calculate average product delivery rate (PDR). According to our research the average PDR for TTPoints is around 25h/TTPoint.

The obvious drawback of early-effort estimation is its low accuracy due to the high level of uncertainty related to the final scope of the project. Still, the results we obtained for effort estimation based on TTPoints are promising [27]. We were able to estimate effort with on-average error (MMRE) at the level of ∼0.26, which was on-average lower by 0.14 to 0.22 than the estimation error for different variants of the Use Case Points method.

Claim 1. Effort estimation based on use cases can provide reasonably good estimates.

5 User Manual Generation

Ongoing research shows that up-to-date project documentation can be beneficial, especially when it facilitates the work of a software development team in an automatic way. This section describes initial research concerning automatic generation of a user manual on the basis of software requirements and other information available in a project. An example of a tool that creates a description of fields used in forms is also presented.

The proposed approach focuses on web applications and static documents (documents that do not allow any interaction with users). It is also assumed that the generated user manual is aimed at IT-laymen, people whose computer knowledge is low.

Results of the conducted research show that on the basis of project documentation a number of descriptions can be generated and the produced elements can be used to create a user manual. Functional requirements in the form of use cases are especially helpful, since this form of representation contains an amount of information which is important from the point of view of end users. For example, it is possible to create a list of functionalities available to end users. Each function can be decorated with a description of actions required to get the desired results and a description of exceptions that may occur. To improve the usefulness of the user manual, the description of available functionality can be enriched by a set of screen shots taken from the running application. To get them in an automatic way one can use acceptance tests. After combining use cases and acceptance tests (such information can be got from tools used for traceability), one can easily obtain images.

The process of enhancement can go further, depending on the available information and the possibility of processing it. An example of additional information provided to end users is the field explanation presented in Figure 5. This figure is generated on the basis of a regular expression used in an application to check whether data provided by a user is a valid number of a credit card or not. Our research shows that regular expressions are sufficient to generate three types of description: an explanation in natural language, a diagram and a set of examples. To present regular expressions in an easy to understand way, a special version of syntax diagrams has been designed [5].

Conjucture 2. Use cases can significantly support generation of a user manual.

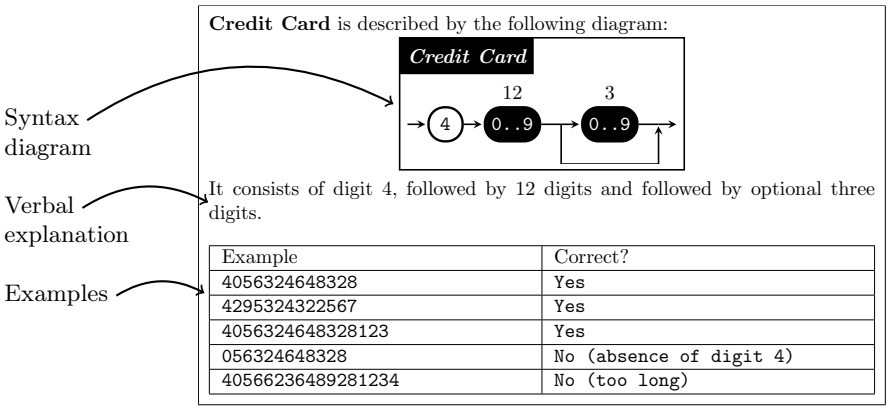

Fig. 5. Field explanation of a credit card number

6 Conclusions

In this paper we have discussed research issues concerning agile requirements engineering. Our findings are as follows:

- Use-cases provide a flexible way of describing functional requirement. They can be presented at several abstraction levels: from extremely concise context diagrams, through use-case diagrams, main scenarios, exceptions, down to the level of alternative steps. Some use-cases can be presented only at the level of use-case diagrams (i.e. lacking further description) while others can also have exceptions and alternative steps specified.
- People are not very good at identifying exceptions (events). Their effectiveness is below 30%. However, from our early experiments it follows that events identification performed by a computer can have effectiveness at the level of 80%. That is a good incentive for further research in this area.
- Non-functional requirements can be elicited in very short brain-storming sessions driven by ISO 25000 quality characteristics. The stability of non-functional requirements elicited in that way for Internet applications was at the level of 80%. It suggests that non-functional requirements are pretty stable and can be collected early.
- The TTPoints method of effort estimation fits use cases well and its average estimation error is below 30%. This is a pretty good result, but further research seems necessary.
- Taking care of requirements specification pays-off, and not only with effort estimates. When functional requirements (in the form of use cases) are complete, then a considerable part of a user manual can be generated. That option should be particularly interesting in Software Product Lines where there are many variants of the same systems and the same user manual.

Acknowledgments. This work has been partially supported by two projects financed by the Polish National Science Center based on the decisions DEC-2011/01/N/ST6/06794 and DEC-2011/03/N/ST6/03016, and by a Poznan University of Technology internal grant 91-518 / 91-549.

References

1. IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998, pp. 1–40 (1998)
2. Adolph, S., Bramble, P.: Patterns for Effective Use Cases. Addison Wesley, Boston (2002)
3. Albrecht, A.J.: Measuring application development productivity. In: Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, pp. 83–92 (October 1979)
4. Alchimowicz, B., Jurkiewicz, J., Ochodek, M., Nawrocki, J.: Building benchmarks for use cases. Computing and Informatics 29(1), 27–44 (2010)
5. Alchimowicz, B., Nawrocki, J.: Generating syntax diagrams from regular expressions. Foundations of Computing and Decision Sciences 36(2), 81–97 (2011)
6. Beck, K., Andres, C.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley Professional (2004)
7. Bjarnason, E., Wnuk, K., Regnell, B.: A case study on benefits and siede-effects of agile practices in large-scale requirements engineering. In: Agile Requirements Engineering Workshop, Agile RE 2011, pp. 9–13. ACM (2011)
8. Cao, L., Ramesh, B.: Agile requirements engineering practices: An empirical study. IEEE Software 25(1), 60–67 (2008)
9. Cleland-Huang, J.: Quality Requirements and their Role in Successful Products. In: IEEE International Requirements Engineering Conf., pp. 361–361. IEEE (October 2007)
10. Cleland-Huang, J., Czauderna, A., Keenan, E.: A persona-based approach for exploring architecturally significant requirements in agile projects. In: Doerr, J., Opdahl, A.L. (eds.) REFSQ 2013. LNCS, vol. 7830, pp. 18–33. Springer, Heidelberg (2013)
11. Cockburn, A.: Writing Effective Use Cases. Addison Wesley, Boston (2000)
12. Cohn, M.: User Stories Applied: For Agile Software Development. Addison Wesley Longman Publishing Co., Inc., Redwood City (2004)
13. Haugen, N.C.: An empirical study of using planning poker for user story estimation. In: Agile Conference, pp. 25–34. IEEE (2006)
14. ISO. ISO/IEC 25010:2011 - Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. International Organization for Standardization, Geneva, Switzerland (2011)
15. Jacobson, I.: Object-Oriented Software Engineering. Addison-Wesley (1992)
16. Jurkiewicz, J., Nawrocki, J., Ochodek, M., Glowacki, T.: HAZOP-based identification of events in use cases. an empirical study. Empirical Software Engineering (2013) (accepted for publication), doi:10.1007/s10664–013–9277–5
17. Karner, G.: Metrics for objectory. No. LiTH-IDA-Ex-9344:21. Master's thesis, University of Linköping, Sweden (1993)
18. Kopczyńska, S., Maćkowiak, M., Nawrocki, J.: Structured meetings for non-functional requirements elicitation. Foundations of Computing and Decision Sciences 36(1), 35–56 (2011)

19. Leveson, N.G., Turner, C.S.: An investigation of the therac-25 accidents. Computer 26(7), 18–41 (1993)
20. Mahnič, V., Hovelja, T.: On using planning poker for estimating user stories. Journal of Systems and Software 85(9), 2086–2095 (2012)
21. Mylopoulos, J., Chung, L., Nixon, B.: Representing and using nonfunctional requirements: a process-oriented approach. IEEE Transactions on Software Engineering 18(6), 483–497 (1992)
22. Nawrocki, J.R., Olek, Ł.: UC workbench – A tool for writing use cases and generating mockups. In: Baumeister, H., Marchesi, M., Holcombe, M. (eds.) XP 2005. LNCS, vol. 3556, pp. 230–234. Springer, Heidelberg (2005)
23. Nuseibeh, B.: Ariane 5: Who dunnit? IEEE Software 14(3), 15–16 (1997)
24. Ochodek, M., Alchimowicz, B., Jurkiewicz, J., Nawrocki, J.: Improving the reliability of transaction identification in use cases. Information and Software Technology 53(8), 885–897 (2011)
25. Ochodek, M., Nawrocki, J.: Automatic transactions identification in use cases. In: Meyer, B., Nawrocki, J.R., Walter, B. (eds.) CEE-SET 2007. LNCS, vol. 5082, pp. 55–68. Springer, Heidelberg (2008)
26. Ochodek, M., Nawrocki, J.: Enhancing use-case-based effort estimation with transaction types. Foundations of Computing and Decision Sciences 35(2), 91–106 (2010)
27. Ochodek, M., Nawrocki, J., Kwarciak, K.: Simplifying effort estimation based on Use Case Points. Information and Software Technology 53(3), 200–213 (2011)
28. OMG. OMG Unified Modeling Language^{TM}(OMG UML), superstructure, version 2.3 (May 2010)
29. Redmill, F., Chudleigh, M., Catmur, J.: System safety: HAZOP and software HAZOP. Wiley (1999)
30. Schwaber, K.: Scrum development process. In: Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications, pp. 117–134 (1995)
31. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall (2001)
32. Van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Fifth IEEE International Symposium on Requirements Engineering, pp. 249–262. IEEE (2001)