Software Quality and Agile Methods

Article · October 2004

DOI: 10.1109/CMPSAC.2004.1342889 · Source: IEEE Xplore

June M. Verner

152 PUBLICATIONS **4,372** CITATIONS

SEE PROFILE

Liming Zhu
The Commonwealth Scientific and Industrial Research Organisation

258 PUBLICATIONS **5,792** CITATIONS

SEE PROFILE

Muhammad Ali Babar
University of Adelaide

403 PUBLICATIONS **8,450** CITATIONS

SEE PROFILE

Software Quality and Agile Methods

Ming Huo, June Verner, Liming Zhu, Muhammad Ali Babar
National ICT Australia Ltd. and University of New South Wales, Australia
{mhuo, jverner, limingz, malibaba }@cse.unsw.edu.au

Abstract

Agile methods may produce software faster but we also need to know how they meet our quality requirements. In this paper we compare the waterfall model with agile processes to show how agile methods achieve software quality under time pressure and in an unstable requirements environment, i.e. we analyze agile software quality assurance. We present a detailed waterfall model showing its software quality support processes. We then show the quality practices that agile methods have integrated into their processes. This allows us to answer the question "Can agile methods ensure quality even though they develop software faster and can handle unstable requirements?"

1 Introduction

Ever since Kent Beck introduced Extreme Programming [1], agile software development has become a controversial software engineering topic. Practitioners and researchers argue about the benefits of it, others are forcefully against agile methods, while others suggest a mix of agility and plan-driven practices [2]. However, the reality is that agile methods have gained tremendous acceptance in the commercial arena since late 90s because they accommodate volatile requirements, focus on collaboration between developers and customers, and support early product delivery. Two of the most significant characteristics of the agile approaches are: 1) they can handle unstable requirements throughout the development lifecycle 2) they deliver products in shorter timeframes and under budget constraints when compared with traditional development methods [3-6]. Many reports support the advantages of agile methods. However, proponents of agile methods have not yet provided a convincing answer to the question "what is the quality of the software produced?" Does agility provide enough rigors to ensure quality, as do traditional development methods, e.g., waterfall model, and if agile methods do provide the same level of quality then how is it achieved?

We now compare the quality assurance techniques of agile and traditional software development processes. Our approach consists of three steps: 1) build a complete outline of the traditional waterfall model including its supporting processes, 2) identify those practices within agile methods that purport to ensure software quality, 3) determine the similarities and differences between agile and traditional software quality assurance techniques. By applying such an approach, we can systematically investigate how agile methods integrate support for software quality within their life cycle.

The rest of the paper is organized as follows: Section 2 presents a short description of waterfall and agile methods to highlight the reasons why the latter have become popular. Section 3 gives a brief introduction to software quality assurance techniques. Section 4 explains why we chose a waterfall approach for our comparison. Section 5 concludes the paper.

2 Waterfall model vs. Agile Methods

Even though, on an abstract level, the waterfall model and agile methods are very different process methods, their actions within the development sequence share some similarities. In this section, we provide a short description of both the waterfall model and agile methods. In 2.3, we present how one short agile release shares similar development activities with the waterfall model.

2.1 Waterfall model

Since the late 60s, many different software development methodologies have been introduced and used by the software engineering community [7].

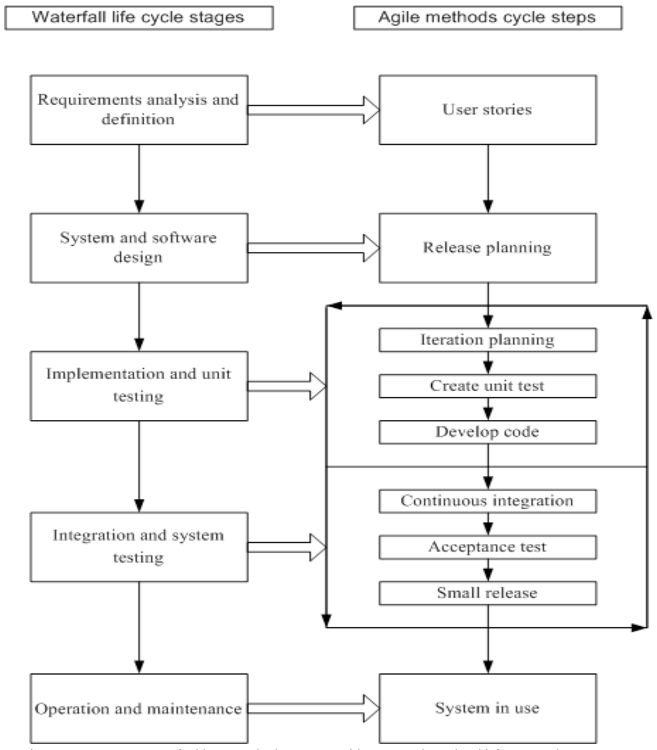
Over the years, developers and users of these methods have invested significant amounts of time and energy in improving and refining them. Owning to continuous improvement efforts, most of the methodologies have reached a mature and stable level. Hence, they are referred as traditional software development methods. Each of the traditional development methods attempts to address different development issues and implementation conditions. Among the traditional development approaches, the waterfall model is the oldest the software development process model. It has been widely used in both large and small software intensive projects and has been reported as a successful development approach especially for large and complex engineering projects [7]. The waterfall model divides the software development lifecycle into five distinct and linear stages. Because it is the oldest and the most mature software development model we have chosen it to investigate its QA process [8]. In addition we chose the waterfall model because the phases in a waterfall development are more linear than other models. This provides us the opportunity to clearly present the quality assurance (QA) processes. In practice, the waterfall development model can be followed in a linear way. However, some stages can also be overlapped. An iteration in an agile method can also be treated as a miniature waterfall life cycle. Despite the success of the waterfall model with large and complex systems, it has a number drawbacks, such as inflexibility in the face of changing requirements, and a highly ceremonious processes irrespective of the nature and size of the project [7]. Such drawbacks can also be found in other traditional development approaches. However, agile methods were developed to address a number of the problems inherent in the Waterfall model.

2.2 Agile Methods

Agile methods deal with unstable and volatile requirements by using a number of techniques. The most notable are: 1) simple planning, 2) short iteration, 3) earlier release, and 4) frequent customer feedback. These characteristics enable agile methods to deliver product releases in a much short period of time compared to the waterfall approach. This brief comparison of the waterfall model and agile methods brings this discussion to our research question, "How can agile methods ensure product quality with such short time periods?" Our research hypothesis is that included in an agile methods development lifecycle, to some degree, are some practices, which offer traditional QA supporting processes.

2.3 One agile release vs. waterfall life cycle

The waterfall development model provides us with a high-level framework and within this framework, are five distinct stages: 1) requirements analysis and definition 2) system and software design 3) implementation and unit testing 4) integration and system testing 5) operation and maintenance [7]. In principle, any stage should not start until the previous stage has finished and the results from the previous stage are approved. The agile approach turns the traditional software process sideways. Based short releases, agile methods go through all development stages a little at a time, throughout their software development life cycle. In one agile release, the steps may not be clearly separated, as they are in a waterfall development model, but the requirements recognition, planning, implementation and integration sequences are the same as in waterfall model. Figure 1 lists a short comparison between the waterfall model and agile methods.



Waterfall life cycle stages
Agile methods cycle steps
Requirements analysis and definition
User stories
System and software design
Release planning
Implementation and unit testing
Iteration planning
Create unit test
Develop code
Integration and system testing
Continuous integration
Acceptance test
Small release
Operation and maintenance
System in use

Figure 1 Waterfall model vs. agile methods life cycle

3 Quality assurance techniques

Since we are concerned with the quality of software produced with the Waterfall model and an agile approach, we investigate quality-centric software development supporting processes. We concentrate on two of the most widely used general quality-focused processes, Software Quality Assurance (SQA) and Verification and Validation (V&V).

"SQA governs the procedures meant to build the desired quality into the products" and V&V is aimed more directly at product quality including intermediate products [8]. These two supporting processes are normally used to support the waterfall model to provide a complete process model. QA techniques can be categorized into two types, static and dynamic. The selection, objectives, and organization of a particular technique depend on the requirements and nature of the project and selection is based on very different criteria [8] depending on the methodology being used.

Unlike dynamic techniques, static techniques do not involve the execution of code. Static techniques involve examination of documentation by individuals or groups. This examination may is assisted by software tools, e.g., inspection of the requirements specification and technical reviews of the code. Testing and simulation are dynamic techniques. Sometimes static techniques are used to support dynamic techniques and vice versa. The waterfall model uses both static and dynamic techniques. However, agile methods mostly use dynamic techniques.

4 Agile methods quality techniques

Figure 2 shows a complete model of a waterfall development with its QA supporting process in diagrammatic form. In the next diagram (Figure 3), we show the agile methods life cycle in diagrammatic form. In 4.2 we address some quality assurance practices used by agile methods.

4.1 Waterfall model with SQA and V&V

The development activities in the Waterfall model include: 1) requirements definition 2) system and software design 3) implementation and unit testing 3) integration and system testing 4) operation and maintenance [7]. Each activity produces well-defined deliverables. Since the deliverables of one activity are input for a subsequent activity, from the theory point of view, no subsequent phase can begin until the predecessor phase finishes and all of its deliverables are signed off as satisfactory.

In Figure 2, the left hand side shows the main waterfall development model and the right its supporting processes. The output from each phase is input to the corresponding supporting phase and will be verified or validated by its supporting process; this output is then sent to the next stage as input.

We use the model shown in Figure 2 as a base for comparison with the QA techniques of agile methods. We explain this further in section 4.4.

4.2 Agile methods with QA

In Figure3, we present a generalized agile method development life cycle. In this diagram, some agile stages normally overlap each other. This makes it difficult to show distinct phases. The generic development sequence is the same as in the waterfall model (Figure 2) however, in Figure 3 the repeated unit cycle is a short release, which does not exit in the normal waterfall model. In Figure 3, the left hand side shows the agile processes main sequence and the right side includes agile practices that have QA ability. There are two major differences between Figures 2 and 3; 1) in agile methods, there are some practices that have both development functionality and as well as QA ability. This means that agile methods move some QA responsibilities and work to the developers. These practices are marked by an underline and are discussed in detail in section 4.3. 2) In an agile methods phase a small amount of output is sent frequently to quality assurance practices and fast feedback is provided, i.e., the development practices and QA practices cooperate with each other tightly and exchange the results quickly in order to keep up the speed of the process. This means that the two-way communication speed in agile methods is faster than in a waterfall development.

4.3 Agile Methods: quality techniques

Agile methods include many practices that have QA potential. By identifying these practices and comparing them with QA techniques used in the waterfall model, we can analyze agile methods QA practices. *System metaphor* is used instead of a formal architecture. It presents a simple shared story of how the system works; this story typically involves a handful of classes and patterns that shape the core flow of the system being built. There are two main purposes for the metaphor. The first is communication. It bridges the gap between developers and users to ensure an easier time in discussion and in providing examples. The second purpose is that the metaphor contributes to the team's development of a software architecture [10]. This practice helps the team in architecture evaluation by increasing communication between team members and users.

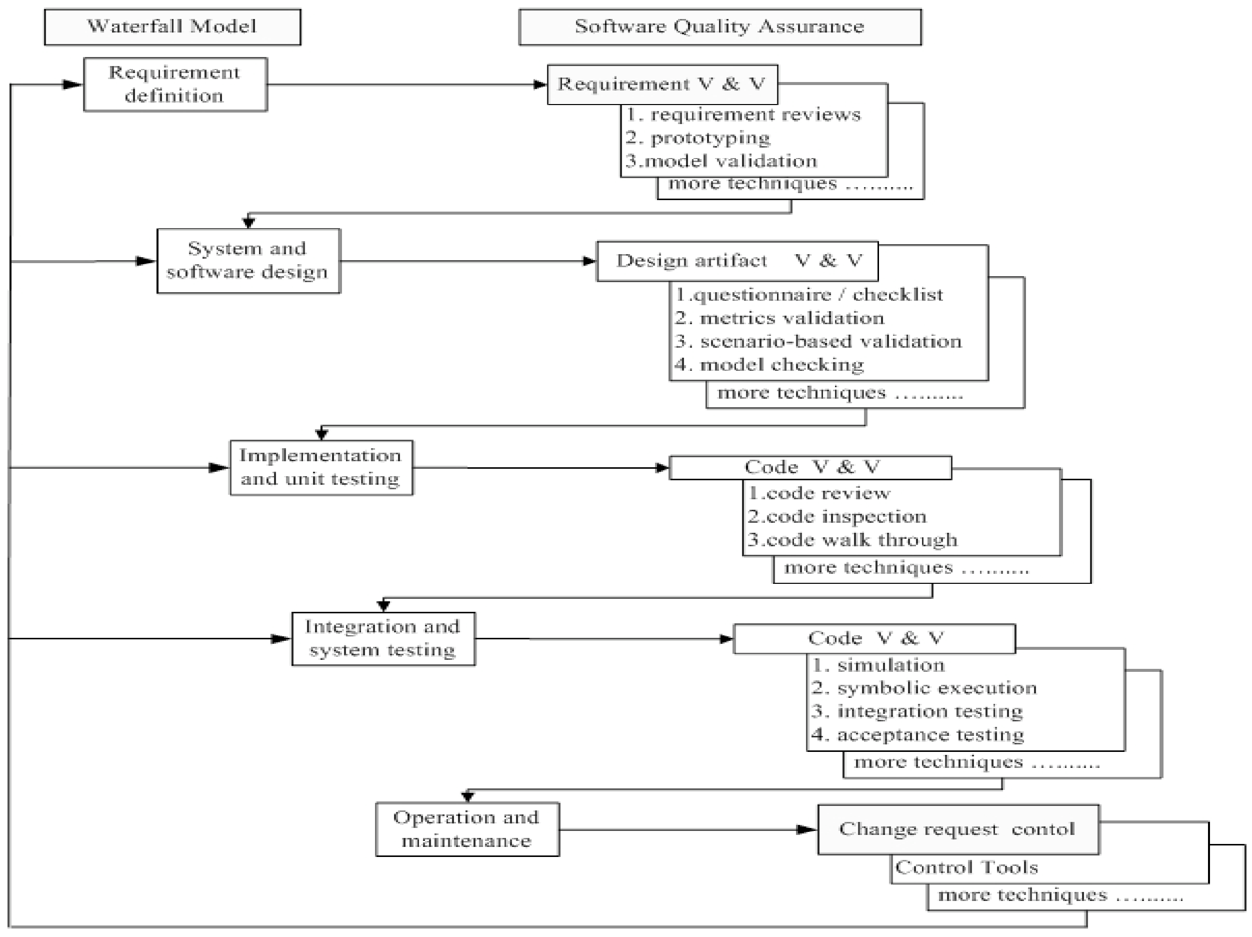


Figure 2 Completed Waterfall Process Model

Having an *On-site customer* is a general practice in most agile methods. Customers help developers refine and correct requirements. The customer should support the development team throughout the whole development process. In the waterfall model, customers are typically involved in requirements definition and possibly system and software design but are not involved as much and do not contribute as much as they are expected to in an agile development. Consequently customer involvement in agile methods is much heavier than in waterfall development. In practice, in a waterfall development, some milestone reviews might be set up and customers will participate, but this kind of customer involvement is less intense than it is in an agile development.

Pair programming means two programmers continuously working on the same code. Pair programming can improve design quality and reduce defects [11]. This shoulder-to-shoulder technique serves as a continual design and code review process, and as a result defect rates are reduced. This action has been widely recognized as continuous code inspection [11].

Refactoring "is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. Its heart is a series of small behavior preserving transformations. Each transformation (called a 'refactoring') does little, but a sequence of transformations can produce a significant restructuring." Because each refactoring is small, the possibility of going wrong is also small and the system is also kept fully functional after each small refactoring. Refactoring can reduce the chances that a system can get seriously broken during the restructuring [12]. During refactoring developers reconstruct the code and this action provides code inspection functionality. This activity reduces the probability of generating errors during development.

Continuous integration, a popular practice among agile methods means the team does not integrate the code once or twice. Instead the team needs to keep the system fully integrated at all times. Integration may occur several times a day. "The key point is that continuous integration catches enough bugs to be worth the cost" [12]. Continuous integration reduces time that people spend on searching for bugs and allows detection of compatibility problems early. This practice is an example of a dynamic QA technique. Waterfall model development integration is done much later and its frequency is much lower than in an agile method development [13].

Acceptance testing is carried out after all unit test cases have passed. This activity is a dynamic QA technique[8]. A Waterfall approach includes acceptance testing but the difference between agile acceptance testing and traditional acceptance testing is that acceptance testing occurs much earlier and more frequently in an agile development; it is not only done once. *Early Customer feedback* is one of the most valuable characteristics of agile methods. The short release and moving quickly to a development phase enables a team to get customer feedback as early as possible, which provides very valuable information for the development team.

We can compare the differences between the SQA from three aspects: 1) many of the agile quality activities occur much earlier than they do in waterfall development, 2) the frequency of these activities is much greater than in the waterfall model; most of these activities will be included in each iteration and the iterations are frequently repeated during development, 3) agile methods have fewer static quality assurance techniques.

Agile methods move into the development phase very quickly. Although this kind of development style renders most separate static techniques on early phase artifact unsuitable, code makes dynamic techniques useful and available very early. Also developers are more responsible for quality assurance compared with having a separate QA team and process. This allows more integration of QA into the development phase. Small releases also bring customer feedback for product validation frequently and requirements verification. The QA techniques for agile methods are based on:

- Applying dynamic QA techniques as early as possible (e.g. TDD, acceptance testing)
- Moving more QA responsibility on to the developer (e.g. code inspection in peer/pair programming, refactoring, collective code ownership, coding standards)
- Early product validation [8] (e.g. customer on site, acceptance testing, small release, continuous integration)

Figure 4 shows the waterfall model and agile development methods life cycles based on time and their available quality assurance techniques. We can see that the dynamic techniques are applied late in a waterfall development when compared with agile development. In an agile development cycle, static and dynamic techniques can both be applied from very early stages.

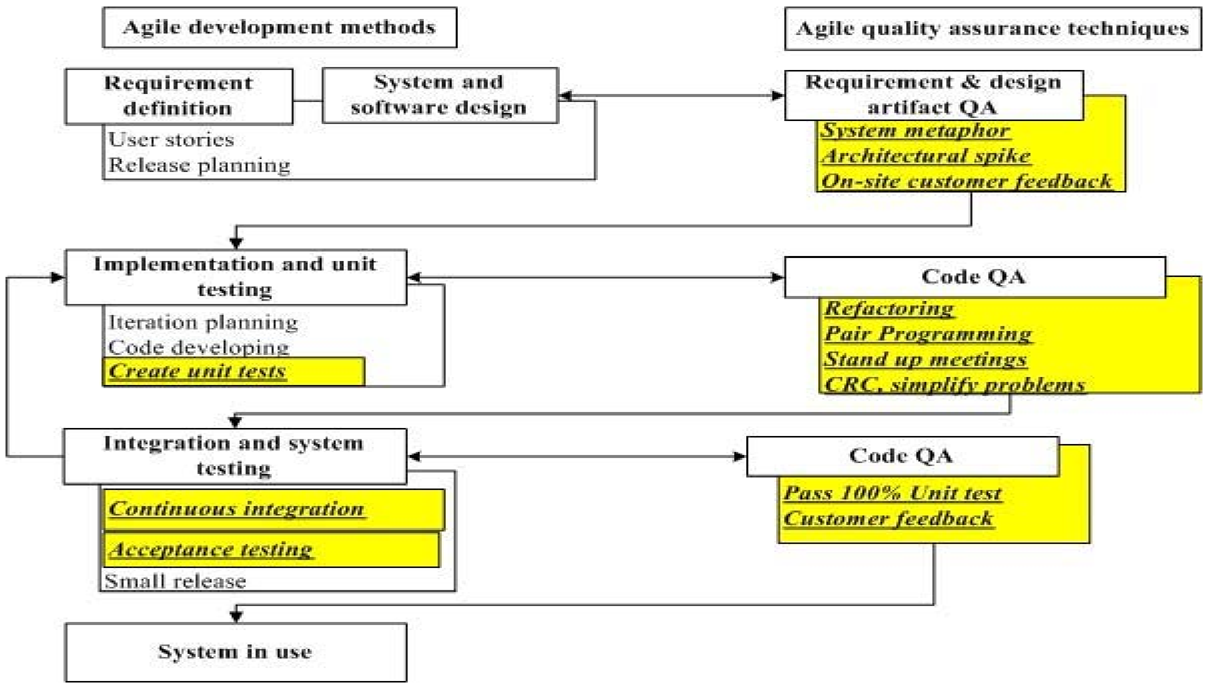


Figure 3 Agile methods and QA [9]

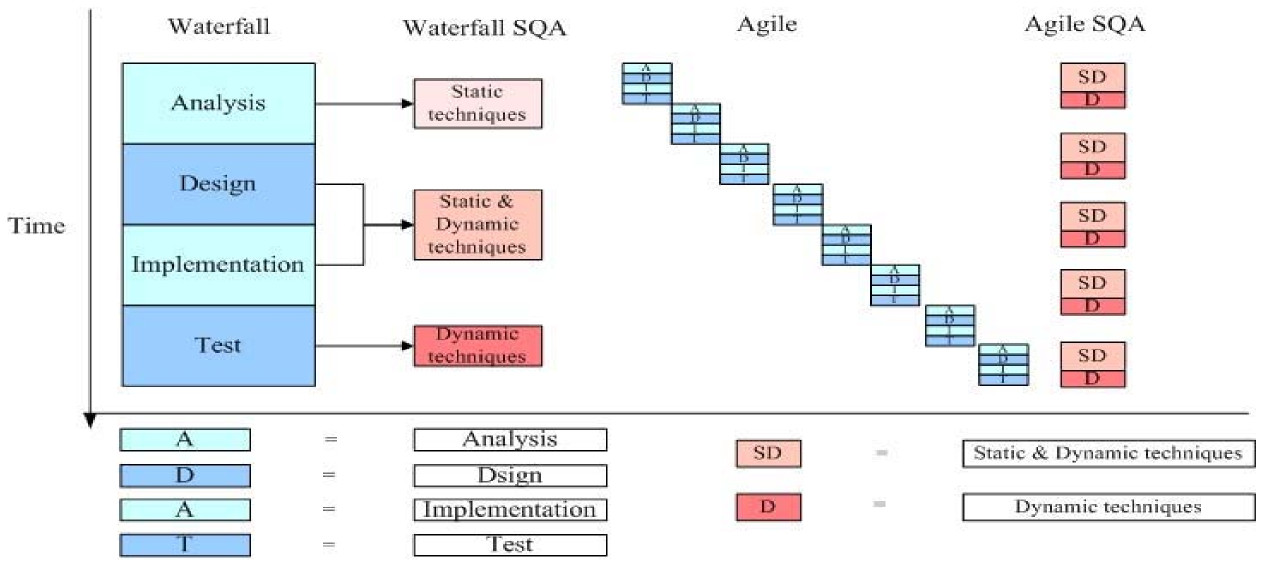


Figure 4 SQA Timeline [14]

5 Conclusion

Even though some agile practices are not new, agile methods themselves are recent and have become very popular in industry. There is an important need for developers to know more about the quality of the software produced. Developers also need to know how to revise or tailor their agile methods in order to attain the level of quality they require. In this paper we have analyzed agile practices' quality assurance abilities and their frequency. The conclusion we draw here is: 1) agile methods do have practices that have QA abilities, some of them are inside the development phase and some others can be separated out as supporting practices 2) the frequency with which these agile QA practices occur is higher than in a waterfall development 3) agile QA practices are available in very early process stages due to the agile process characteristics. From this analysis, we identified some issues for which development criteria might be desirable. According to the process quality a team require and time they have available they can tailor agile practices.

However, is difficult, sometimes even not realistic to compare the software quality resulting by the use of a waterfall model with agile methods because their initial development conditions, especially the cost, are not comparable.

6 References

[1] K. Beck, *extreme programming eXplained: embrace change*. Reading, MA: Addison-Wesley, 2000.

[2] B. Boehm and R. Turner, "Using risk to balance agile and plan-driven methods," *Computer*, vol. 36, pp. 57-66, 2003.

[3] J. Grenning, "Launching extreme programming at a process-intensive company," *Software, IEEE*, vol. 18, pp. 27-33, 2001.

[4] O. Murru, R. Deias, and G. Mugheddue, "Assessing XP at a European Internet company," *Software, IEEE*, vol. 20, pp. 37-43, 2003.

[5] J. Rasmussen, "Introducing XP into Greenfield Projects: lessons learned," *Software, IEEE*, vol. 20, pp. 21-28, 2003.

[6] P. Schuh, "Recovery, redemption, and extreme programming," *Software, IEEE*, vol. 18, pp. 34-41, 2001.

[7] I. Sommerville, *Software engineering*, 6th ed. Harlow, England ; New York: Addison-Wesley, 2000.

[8] A. Abran and J. W. Moore, "Guide to the software engineering body of knowledge : trial version (version 0.95)." Los Alamitos, CA: IEEE Computer Society, 2001.

[9] Extreme Programming: A gentle introduction, http://www.extremeprogramming.org.

[10] How Userful Is the Metaphor Component of Agile Methods? A Preliminary Study,

[11] A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming," in *Extreme Programming examined*, G. Succi and M. Marchesi, Eds. Boston: Addison-Wesley, 2001, pp. xv, 569 p.

[12] M. Fowler, "Information about Refactoring," 2004.

[13] Continuous Integration, http://www.martinfowler.com/articles/continuousIntegration.html.

[14] K. Beck, "Embracing change with extreme programming," *Computer*, vol. 32, pp. 70-77, 1999.