A Case Study on the Adoption of Measurable Agile Software Development Process

Martin Iliev, Iva Krasteva, and Sylvia Ilieva

Faculty of Mathematics and Informatics, Sofia University,
5 James Bourchier Blvd, Sofia 1165, Bulgaria
mg_iliev@abv.bg, iva.krasteva@rila.bg, sylvia@acad.bg

Abstract. Agile methodologies for software development meet the challenges of the current highly dynamic and competitive business environment. The aim of this case study is to improve existing software development process in a project for the public administration, following the basic principles of agile methodologies. Appropriate metrics for continuous evaluation of the process are introduces to help evaluating and improving the methodology. The main objectives of the new methodology are to improve communication with customers, to improve communication among different distributed teams and inside the teams, and to continuously evaluate the way software is developed through selection and usage of software metrics. The paper presents the results of methodology adoption in two subsequent iterations of a real project.

Keywords: measurements, software process, agile software development.

1 Introduction

Agile methodologies for software development appeared as a response to the heavy weight processes in which process is driven by documentation and bearing in mind the idea on how to minimize future change. Current business environment and the technologies change so fast that the classic methodologies are not able to respond to these new challenges [1]. The aim of this case study is to improve existing software development process, following the basic principles of agile methodologies, and to introduce appropriate metrics for continuous evaluation of the process. Software metrics are used to measure specific attributes of the process or produced product, to analyze defects and present a verification of good practice. We have identified and measured several metrics that are valuable for this particular case study. Those metrics are of interest for the different roles of developers. Architects and developers are interested in priority to the quality of the code and its design, while managers are interested mainly in metrics which evaluate time needed to produce unit of functionality. The proposed methodology is based on the results and experience of well known agile development methods. The goal is to identify and apply practices and techniques that are suitable for carrying out specific software project for public administration. The main objectives of the new methodology are to improve communication with customers, to improve communication among different distributed teams and inside the teams, and to continuously evaluate the way software is developed through selection and usage of software metrics.

The paper consists of six sections. First, it makes an overview of the proposed methodology, its design as well as the objectives and requirements for it. In section 3 metrics that will help us to study the results of applying the methodology are introduced. Section 4 presents the case study and the way methodology was implemented. Results of the implementation are analysed in section 5. Section 6 concludes the paper.

2 Design of the New Methodology

Current section makes an overview of the proposed agile methodology. First, the requirements and prerequisites for methodology design are specified which are based on the shortcomings of the existing process and particular project characteristics. Then the design of the new methodology, which satisfies the identified requirements, is described.

Different projects have different requirements for the development process and characteristics of the process are closely dependent on the specifics of the project and the capabilities of the organization which implements this process. The following requirements should be taken into account when new methodology is specified:

- quality of the developed software product should be improved;
- development should be predictable for the client;
- there should be direct communication between the client and the developers;
- communication within the team and between different teams should be improved;
- effective control over the process of development should be imposed.

The new methodology is designed in accordance to the requirements of subsection 2.1 and is based on the 4CC framework (Four Cycles of Control) proposed by Rautiainen [2]. The framework allows the combination of business and process management through four cycles of control as shown on Figure 1. Fundamental concepts of the proposed process are cycles of different management and development activities. Each cycle has a different time, purpose and detail that characterized it. Proper arrangement and organization of cycles is essential for the implementation of the direct control of the process of development and continuous learning from previous experience. The four cycles of control in this framework are strategic management, release management, iteration management and daily rhythm.

Strategic management cycle forms the overall vision for the project. During this cycle a time frame for the project consistent with the technology and market requirements are prepared. It is focused on long-term tasks such as launching a product or terminating or freezing work on a product. The *release management* is associated with the development of a new product or new version of a product. It develops vision for the product which may be a synthesis of customer feedback, market studies, product roadmap and more. The output from this cycle is a new product or new version of the product. During the *iteration management* cycle the next artifact (documentation or functionality) as set out in the roadmap of the product is prepared. Iteration management has a time-frame of about one month.

This cycle corresponds with Scrum's one month iteration. The *daily rhythm* cycle is used to manage and motivate the development against the objectives of the sprint. Work of individual members and teams involved in development are synchronized and the product is integrated and tested constantly. *Supporting activities* are ongoing activities that guide development and continue during all cycles. Their main goal is to provide effective methods and tools necessary for configuration management, requirements management, tracking defects, daily builds and testing.

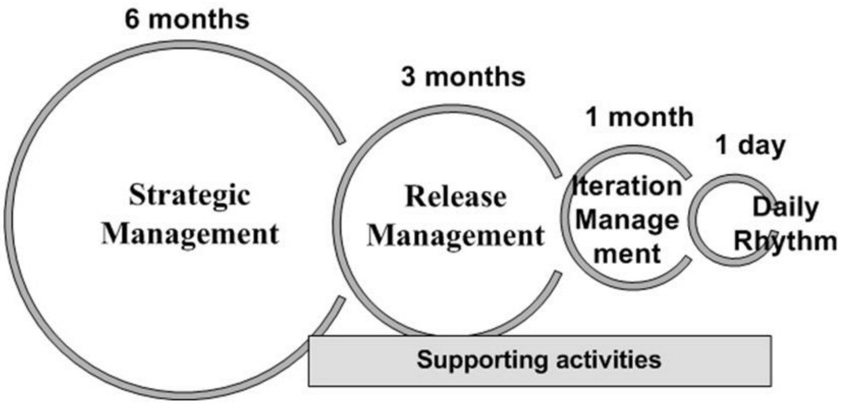

Fig. 1. Cycles of control.

3 Metrics

According to DeMarco [3] "you cannot control what you cannot measure". Or said otherwise without evolution software metrics and software process improvements are impossible [4]. Software metrics are used to measure specific attributes of the process or produced product, analyzing defects, verification of good practice [4]. Below are listed and detailed metrics that are most appropriate for the case study project product. Those metrics are of interest for the different roles of developers. Architects and developers are interested in priority to the quality of the code and its design, while managers are interested mainly in metrics which measures time needed to produce unit of functionality.

The idea of any software project is to produce software that works and to the maximum extent covers the requirements set out in the beginning of the project [5]. We have used the above idea and created a metric which is called *Running Test Functionality* (RTF). The RFT metric can be defined as follow:

- Software product is broken down into named features (requirements, stories), part of the system to be developed
- For each of the features there are one or more automated acceptance tests. If they work it means that the functionalities are implemented correctly.
- The RTF metric shows, at any moment in the project how many features are passing their acceptance tests.

The agile methodologies have one distinguishing feature –code refactoring [7]. Source code refactoring means changing the programming code and software design, without any changes in end user functionality of the software. For example, in XP at the end of each iteration source code refactoring take place. The main objective is to provide basic quality attributes for the product. Therefore refactoring cycle is critical for flexible software development. In support of flexible methodology and especially in the refactoring are used programming code software metrics [6]. Our goal is to combine the refactoring and software metrics, it will suggest answers to the following questions:

- When is the most accurate time for refactoring?
- What is significance of each refactoring step and its qualitative effect?
- Which are the side effects of refactoring?

These metrics should provide indicators of the need for certain steps for refactoring. The values of these metrics must be easily interpreted and have explicit semantic. The results should be trigger for refactoring. We use the following software metrics which are based on different source code metrics:

- Coupling (Efferent Coupling)
- Complexity (Cyclomatic Complexity)

Software metric *Efferent Coupling* indicates the following:

- good encapsulation;
- high level of abstraction;
- good opportunity for reuse;
- easy extensibility;
- low development costs;
- low maintenance costs.

A low *Cyclomatic Complexity* of the developed code leads to:

- low maintenance costs;
- collective code ownership;
- easy to test and produce good code coverage results.

4 Case Study Description

This project represents the realization of a distributed system. Each part in this distributed system is connected with each other through common transport environment. The development is separated in the following modules: Core module which deals with complete processing of messages, Business process module implements processes defined by the client, Web module implements user interface.

The project team organization has following roles:

- Project manager. Responsible for the successful implementation of the project. Project manager coordinates with the client deadlines, iterations, prepare development plan, requirements analysis
- Business analyst team. Responsible for the analysis and specification of the of clients processes.

- Software architect. Responsible for overall technical design and implementation of the system. Perform coordination between the teams.
- Developers. Developer teams are divided according to the modules described above. Each team consists of 4 to5 developers. Each team has a person responsible for the design and implementation – team leader.

The existing software process in the company has three main phases - The initial phase of design and specification, construction phase and transition and adoption phase. During the initial phase is planned development of the whole project which includes – iteration planning, team roles and team organization. Use Case model is being prepared for the main scenarios. This first phase includes also design and overall architecture of the system. The construction phase consists of one or more iterations in which happens the actual construction of the system. This phase ends with a test period of about a week. In the transition and adoption phase the customer provides tests to cover the requirements - acceptance tests. After their successful project cover is ready to be delivered to the client.

The implementation steps follow the four cycles of control described in chapter 2. In strategic management client specify time frame for project execution, the development teams was formed based on each developer technical expertise. During release management cycle were defined all project iterations. In practice this is scrum sprints with duration of one month. The key priorities are being defined. According to the type they might be business priorities or technical priorities. Iteration is implemented in a few steps which include planning session, functional construction, testing strategy implementation, sprint review. Daily activities include design, coding, testing and metrics gathering. Support activities is a set of practices are essential for the tasks of daily rhythm and the overall success of each sprint. They include – task management, defects management, daily builds, automated testing.

The process implementation is observed during first two project iterations. *First iteration* began according to process design – the key priorities were defined. The main business priority was to deliver to the client a small core functionality of implemented system during this first iteration. The technical priority was to choose best software technologies which will suite best on our customer requirements. To evaluate the best process server, ORM framework and web framework two small pilot projects were made.

During the *second iteration* some important improvements were made. In the first iteration code coverage was up to 50 percents while during the second iteration the code coverage increased to 70 percent in order to satisfy customer requirements for “fully tested” software system. In first iteration we had scattered and fragmented tasks, while in second iteration were taken the approach of creating bigger tasks which consists of subtasks. In consequence it is much easier for testers to associate certain defect with a task. It is easy to see how many and what faults have occurred for a task.

5 Analysis of the Results

As shown in Figure 2 (C1 denotes first iteration and C2- the second) RTF curve decrease at the beginning of the C2. This means that there is functionality and tests which are no longer running. Reason for this decline is the practice at the beginning of each iteration to have a period of two to five days to inspect and refactor functionality developed in the previous cycle. This decline has the following effects on the process of development:

- Discussion and refactor already implemented functionality. This is very important to improve the quality of design and implementation.
- After each iteration refactoring work decreases. This is as a result of performed refactoring in the previous iterations. There is gradual straightening of the RTF curve.

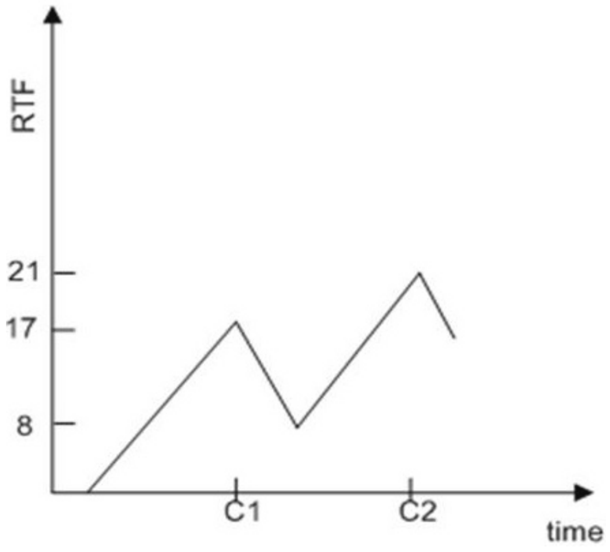

Fig. 2. RFT metric for two iterations.

The measurements for the Efferent Coupling (EC) show that there are 14 classes in first iteration with EC values bigger than 25 (Figure 3), while second iteration has 18 classes with value for EC bigger than 25 (Figure 4). Classes with values for EC exceeding 25 are candidates for refactoring. In this case, were taken the following actions:

- Discuss decomposition of this class into several classes with smaller values of EC.
- Classes which are planned to be refactored are reafactored. In some cases, this step may be performed in next iteration in development. This is eligible for cases where it requires a significant revision of the code and this may cause implementation delay of the functionality planned for the current sprint.

Particularly in the case, we can say that the increase in the value of the EC in second iteration in comparison to the first iteration is due to the increased volume of code between the two cycles of development.

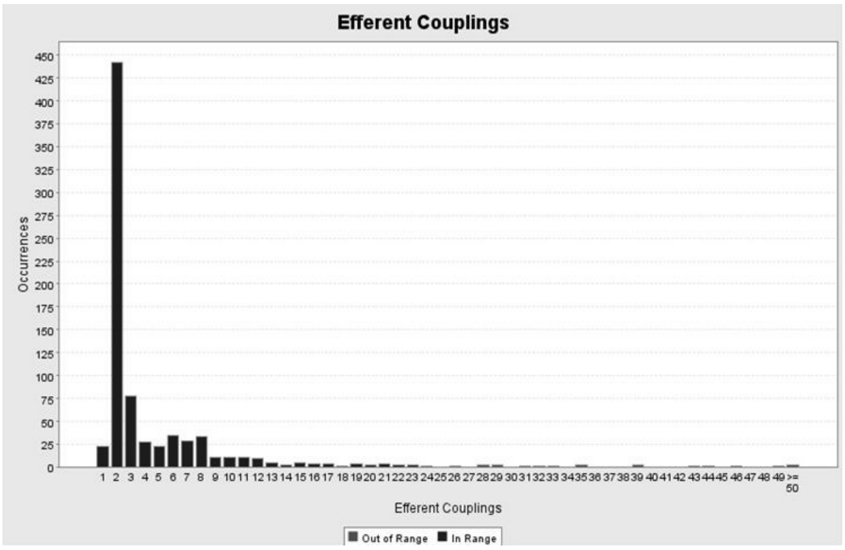

Fig. 3. Efferent Coupling metric for the first iteration.

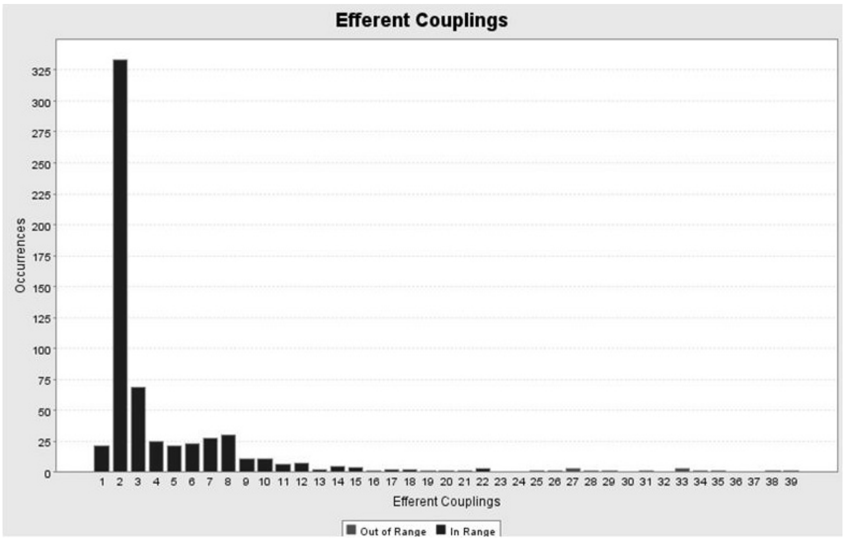

Fig. 4. Efferent Coupling metric for the second iteration.

Overall, the number of high complexity methods implemented in both iterations is not great, but when metrics indicate a method for passing permitted complexity leads to the creation of a normal priority task whose purpose is the following:

- Make revision to the implementation and to try to reduce complexity.
- Write more unit test methods to increase test coverage of the methods with high complexity.

6 Conclusion

As a result of proposed methodology the development process became open to the client. The client feedback after each iteration was very useful in order to improve user functionality of the system. The incremental approach in the implementation of the product let us to demonstrate stable and operating software. Practices introduced in the process of development, testing of the finished functionality, collecting and analyzing various metrics led to improvements in the technical implementation of the project. The implementation of new functionality becomes a predictable and transparent for the client process.

The topic of flexible methodology is quite varied and extensive. Work can be complemented and extended by the description of specific steps in the preparation of the team and organization for migration to agile progress. It could be explored how flexible working is the process affect the overall process of development in the organization and how the different actors in development have contributed to it.

The proposed methodology can be extended with the introduction of new practices that will make it suitable for use in projects with different size and complexity. From the two iterations examined in Section 5 development shows that the uptake and use of a methodology is an open process in which the methodology has gradually improved and adapted to specific conditions.

Acknowledgements. This work is partly funded by the Bulgarian Ministry of Education and Science.

References

1. Highsmith J., Orr K., Cockburn A., “Extreme programming”, in: E-Business Application Delivery, Feb. 2000, pp. 4–17. Available at : http://www.cutter.com/freestuff/ead0002.pdf.
2. Rautiainen , K., A Framework for Managing Software Product Development. Engineering Management Journal, Vol. 14, No. 2, June 2002, pp. 27-32.
3. T. DeMarco, Controlling Software Projects: Management, Measurement & Estimation, Prentice-Hall, 1982.
4. Grady, G.B. Practical Software Metrics for Project Management and Process Improvement. Englewood Cliffs, NJ: Prentice Hall PTR, 1992
5. Jeffries R., http://www.xprogramming.com/xpmag/jatRtsMetric.htm, 2004
6. Kunz M., Reiner R. Dumke, Niko Zenker “Software Metrics for Agile Software Development”, Software Engineering Group University of Magdeburg, Germany
7. Kent Beck, (1999) “Extreme Programming Explained: Embrace Change”, Addison-Wesley Professional