A Case Study to Enable and Monitor Real IT Companies Migrating from Waterfall to Agile

Antonio Capodieci, Luca Mainetti, and Luigi Manco

Department of Innovation Engineering,
University of Salento, Monteroni Street, 73100 Lecce, Italy
{Antonio.capodieci,luca.mainetti,luigi.manco}@unisalento.it

Abstract. Agile development methods are becoming increasingly important to face continuously changing requirements. Nevertheless, the adoption of such methods in industrial environments still needs to be fostered. Companies call for tools to keep under control both agility and coordination of IT teams.

In this paper, we report on an empirical case study aiming at enabling real companies migrating from Waterfall to Agile. Our research effort has been spent in introducing 11 different IT small and medium-sized enterprises to Agile, and to observe them executing projects. To have a common evaluation framework, we selected a set of 61 metrics, with the purpose of measuring the evolution towards Agile. We provide readers with empirical data on two categories of companies' feedbacks: (i) the metrics they considered to be useful beyond the theoretical definitions; (ii) the tools they integrated with existing development environments to collect data from metrics, and evaluate quantitative improvements of Agile.

Keywords: Agile, Software Metrics, Software Engineering, Waterfall, Migration towards Agile.

1 Introduction

Producing software artefacts is a factory in which technologies and productive methodologies are involved in the production chain to create an output. In recent years, there has been a steady transformation of the software development methodologies, from the classical type, such as Waterfall, to the newer Agile ones. Such a trend is due to the several benefits introduced by these methods, such as increasing customer and developer satisfaction, decreasing effort spent (in terms of cost and time) in marketing under uncertain conditions, flexibility, highly iterative development with a strong emphasis on stakeholder involvement, a general reduction in the defect density of the code, adaptability to changeable requirements and complex specifications, iterative development on smaller size projects, and so forth.

The present paper describes a study related to the adoption of Agile methods by some SMEs[1] belonging to the Apulian IT District, an Italian Industrial District,

[1] Small and medium enterprises (SMEs): companies whose personnel numbers fall below certain limits.

founded in 2009. The District is composed of 94 Apulian software companies, four research centres and universities, and 11 associations and trade unions, with a total of more than 4,000 employees.

The presented study is part of the SMART project (Strategies, Methodologies and technologies for Agile Review and Transformation), funded by the Apulia Region Government. The SMART project involves 11 enterprises, two universities and two research centres, all belonging to the District. The project mission is to create a methodological framework to guide and monitor the adoption of Agile methods within projects already started with classical software engineering techniques or without any methodological approach in their development. Agile methods precisely describe the steps and actions to be undertaken with respect only to projects started from scratch [1]. SMART, therefore, tries to extend Agile methods and to overcome their gaps, working on a set of experimental software projects provided by the involved companies.

The study presented in this paper concerns the first stage of the SMART Project. In this phase, we have selected a wide set of software metrics in order to monitor and to successively analyse the transformation from classical development to Agile methodologies in the involved experimental projects. These metrics provide a complete control dashboard for a software project developed with Agile methods and promote the adoption of Agile practices.

Moreover, further refinements of the metrics framework were based on the needs of the enterprises involved in SMART. We will show how the business needs have conditioned the metrics selection and how they have contributed to the synthesis of an easy-to-use, minimally invasive and efficient monitoring system for software development projects. The enterprises' intervention in the metrics selection has resulted in a monitoring system transparent to the developers and suitable for coexistence with the business processes.

So, with this study we achieve two goals: providing a complete list of software metrics that best fit Agile methodologies and facilitating the adoption of the methods by real companies. In this way, we can support the transition process among development methods by monitoring metrics and provide a broad framework, which needs to be tailored to specific situations.

Selecting a broad-spectrum metrics set, we have produced a general-purpose monitoring framework for Agile software projects, flexible and adaptable to different kinds of managerial and technological conditions.

In the next section we describe the context of the study and the adopted research method. We analyse the systematic approach used to select the metrics according to the enterprises' needs and the research environment. In Section 3 we show the selected metrics. Section 4 describes the role of the enterprises in the selection of the metrics and the tools useful in retrieving the measurements. Here, we analyse their feedback in response to the metrics' submission and the resulting metrics framework changes. In Section 5 we present the related work.

2 Research Context

The 11 SMEs linked to the SMART project produce a global sales volume of about 32 million euros (updated 31st December 2010) and provide direct employment for more than 600 workers. Their tasks deal with ICT in both public and private fields, in many countries around the World. So, their business turnover and employment commitments take on a remarkable relevance among the Apulian IT District.

Although well connected and integrated within the Italian ICT business environment, the enterprises involved in the study lack development approaches and methodologies, as well as the specific skills and experience necessary to perform a migration towards modern development practices. For these reasons, these enterprises still use classical but outmoded development methodologies or do not use any well-defined software production strategy and this deficit represents a limit on the current time-to-market and product quality needs. Despite that, they represent an ideal platform to introduce Agile methodologies, because they work on projects conducted by small teams, tightly connected with customers and subject to variable conditions and requirements.

Here-hence the origin of the SMART project. It aims to provide the involved enterprises with the necessary theoretical and practical competencies in the methodological migration by universities and research centres intervention. The experimental projects used as the base for the metrics synthesis were started with a certain amount of time and they differ from each other in the involved software technologies, requirements, and team compositions. So, they spawned a very heterogeneous research background.

2.1 Research Method

The search for the metrics was based on a systematic research approach, necessarily adapted to the background environment described in the previous section: (i) the training of the companies on Agile methods to achieve vocabulary and know-how alignment; (ii) analysis of the metrics in the scientific literature, with emphasis on those specific to the transition from Waterfall to Agile; (iii) focus groups for submission of the metrics to the partners; (iv) collection of the feedbacks from the partners by questionnaire; (v) metrics refining. The analysis of the state of the art aimed to discover the work relating to the monitored introduction of the Agile methodologies in different software factories. The result of the search was a set of metrics comprising both by a certain number of classical metrics and by several metrics closely suited to Agile methods. The business partners reviewed the selected metrics to prevent the selection of hard-to-retrieve metrics and to finally select really adoptable metrics from the enterprises' point of view. This allowed us to refine the initial set of metrics in order to strike a balance between a solid scientific metrics framework and a really applicable monitoring system. First of all, several meetings and workshops were organized to promote information sharing and to encourage the acceptance of the metrics. Secondly, the partners gave feedback, in the form of answers to questions, such as thaose shown in Table 1.

Table 1. Metrics acceptance questionnaire

Question	Answer values
Metric understanding	From 1=not at all, to 5=completely
Willingness to collect data	From 1=not at all, to 5=completely
Ease of access to data collection tools	From 1=not at all, to 5=completely
Data collection scheduling	Each task, sprint, release, etc.
Comments	Open comments about the metric

3 The Selected Metrics

In this section, we analyse the assembled metrics framework. The selection of the metrics has been based on two main goals:

- Monitoring the migration from sequential software development methods, such as Waterfall, to Agile methods;
- Promoting the practices strictly related to Agile methods, especially Test-Driven Development (TDD) and Refactoring.

Therefore, the measurements collection activity enables the adoption of Agile methodologies, promoting the use of Test-Driven Development and Refactoring practices. In this paper, we do not give the definition for each metric. The definition are referenced in the bibliography. We point out especially to metric classification and contextualization.

3.1 Metrics Classification

As stated in [2], the broad metrics classification includes items such as product, process, objective and subjective, direct and indirect, resource and project metrics. Taking inspiration from the aforementioned article, we use a more restricted categorization to avoid class intersections in the metrics classification, as shown in Table 2.

Table 2. Metrics classification

Classification Set	Metrics categories
Product Metrics	Product size; product complexity; derived product complexity; object oriented; usability; code coverage; quality; refactoring; continuous integration.
Process Metrics	Management metrics and life cycle metrics
Resource Metrics	Personnel metrics (effort metrics, etc.), software metrics and hardware metrics, performance metrics
Project Metrics	Cost, time, quality, risk metrics

3.2 Metrics Definition

In this section we introduce the selected metrics. Each of them is well defined in the literature and here they are only gathered according to the classification specified in the previous paragraph. The only exception is represented by the quality metrics, reworked by us to adapt them to our purposes, starting from their definition.

Product Metrics. The metrics are categorized by the subsets specified in the Table 2.

Product Size. The selected product size metrics are:

- **Logical Lines Of Code (LLOC);**
- **Test Case Point analysis (TCP)** [3–5]**;**
- **Number of classes;**
- **Number of abstract classes/interfaces;**
- **Abstractness (A)** [6]**;**

In the literature there are several different product size metrics. Therefore, we have chosen the most useful from the point of view of the Agile methods. For example, we have selected the Test Case Point analysis instead of the more common Function Point Analysis (FPA), since the first fits better the problems concerning the TDD paradigm and promotes the use of this important practice.

Product Complexity

- **Cyclomatic Complexity (v(G));**
- **Halstead Complexity (V).**

Derived Product Complexity. The metrics below extend the Cyclomatic definition:

- **Essential Complexity;**
- **Integration Complexity;**
- **Design Complexity.**

Object Oriented. The metrics are classified according to four of the basic principles of the object-oriented technique, namely coupling, cohesion, inheritance and polymorphism [6]:

Coupling

- **Afferent Couplings (Ca);**
- **Efferent Couplings (Ce);**
- **Coupling Between Objects (CBO);**
- **Instability (I).**

Cohesion

- **Lack Of Cohesion of Methods (LOCM).**

Inheritance

- **Number of Overridden Methods (NORM);**
- **Depth of Inheritance Tree (DIT);**
- **Number Of Children (NOC);**
- **Specialization Index (SIX).**

Polymorphism

- **Weighted Methods per Class (WMC);**
- **Response For Class (RFC).**

Usability. The selected usability metric is based on MiLE+, a framework that enables a rigorous evaluation of the software application's usability [7]. It aims to analyse the whole elements involved in the user interaction with the application.

The framework proposes two types of application inspection:

- **Technical Inspection (TI):** it is based on some heuristics, which evaluate the quality of design and highlight the implementation defects from the point of view of the designers and work personnel.
- **User Experience Inspection (UEI):** checking the usability aspects strongly related to the user needs. In the UEI several usability scenarios are generated and each of them is divided into different tasks. For each task various heuristics evaluate the quality of the interaction between the user and the content of the application. The UEI allows to recognize in advance the potential problems that may arise when a user uses the application.

Code Coverage. Literature provides well-known unit test coverage criteria [8]:

- **Statement Coverage;**
- **Branch/Decision Coverage;**
- **Boolean/Condition Coverage;**
- **Loop Coverage;**
- **Path Coverage.**

Quality. The quality metrics here proposed are based on the amount and the criticality of the defects found during the development phase. In drawing up them, we have taken into account the difference between the *reworking* and *refactoring* phases.

The reworking phase concerns *an improvement of the external quality of the software*, not ensuring the functional equivalence between the code preceding the change and the next. In reworking, code change is due to its critical defects or to requests for improvements and extensions for already implemented functionalities (a sign of the lack of understanding between the team and the stakeholders).

In contrast, the refactoring phase aims at *improving the internal structure* of the software, without changing the external behaviour [9]. Refactoring procedures are intended to make the software more slender, lighter, and elegant, so as to ensure the maintainability, reusability and performance. Starting from these differences, we have selected the quality metrics from the scientific literature (i.e. [10]), adapting them to the demands imposed by Agile methods. Accordingly they are classified as reworking, refactoring and quality parameters metrics. The latter are quality expressions derived from the union between the first ones..

Reworking

- **Critical Defects (SCO_1[2]):** number of critical defects;
- **Improvement Requests (SCO_2):** number of requested improvements and extensions for already implemented functionalities;
- **Open Reworks (RW_O):** cumulative defective LLOC due to SCO_1 and SCO_2, not yet fixed;
- **Closed Reworks (RW_C):** cumulative LLOC related to fixed SCO_1 and SCO_2;
- **Rework Effort (RW_E):** cumulative effort to fix spent to fix SCO_1 and SCO_2;
- **Total Reworks (RW_T):**

$$RW_T = RW_O + RW_C \tag{1}$$

- **Rework Ratio (RW_R)**

$$RW_R = \frac{RW_T}{LLOC} \tag{2}$$

- **Rework Backlog (RW_B)**

$$RW_B = \frac{RW_C}{LLOC} \tag{3}$$

- **Rework Stability (RW_S)**

$$RW_S = RW_T - RW_C \tag{4}$$

- **Rework Effort Ratio (RW_{ER})**

$$RW_{ER} = \frac{RW_E}{TotalEffort} \tag{5}$$

Refactoring

- **Normal Defects (SCO_3):** number of non-critical defects;
- **Open Refactors (RF_O):** cumulative defective LLOC due to SCO_3, not yet fixed;
- **Closed Refactors (RF_C):** cumulative LLOC related to fixed SCO_3;
- **Refactor Effort (RF_E):** cumulative effort spent to fix SCO_3;
- **Total Refactor (RF_T):**

$$RF_T = RF_O + RF_C \tag{6}$$

[2] Software Change Order.

- **Refactor Ratio (RF$_R$)**

$$RF_R = \frac{RF_T}{LLOC} \tag{7}$$

- **Refactor Backlog (RF$_B$)**

$$RF_B = \frac{RF_C}{LLOC} \tag{8}$$

- **Refactor Stability (RF$_S$)**

$$RF_S = RF_T - RF_C \tag{9}$$

- **Refactor Effort Ratio(RF$_{ER}$)**

$$RF_{ER} = \frac{RF_E}{TotalEffort} \tag{10}$$

Combining the shown quality metrics, we can obtain a new set of quality parameters depending on both the rework and refactor procedures. In some of them, an appropriate weight $0 \le \alpha \le 1$ gives more or less importance to the two phases, providing different perspectives of the quality parameters:

Quality parameters

- **Number of SCOs (N):**

$$N = SCO_1 + SCO_2 + SCO_3 \tag{11}$$

- **Modularity (Q$_{MOD}$):** this value identifies the average broken LLOC per SCO, which reflects the inherent ability of the integrated product to localize the impact of change. The best case should ensure that SCOs are written for singular source changes:

$$Q_{MOD} = \alpha \frac{RW_T}{SCO_1 + SCO_2} + (1-\alpha)\frac{RF_T}{SCO_3} \tag{12}$$

- **Changeability (Q$_C$):** this value reflects the ease with which the products can be changed. While a low number of changes is generally a good indicator of a quality process, the magnitude of effort per change is sometimes even more important:

$$Q_C = \alpha \frac{RW_{ER}}{SCO_1 + SCO_2} + (1-\alpha)\frac{RF_{ER}}{SCO_3} \tag{13}$$

- **Maintainability (Q$_M$):** this value identifies the relative cost of maintaining the product with respect to its development cost by relating the effort ratio with the stability parameter. A value of Q$_M$ much less than 1 would tends to indicate a very maintainable product, at least with respect to development cost. Since we would intuitively expect the maintenance cost of a product to be proportional to its development cost, this ratio provides a fair normalization parameter for comparison between different projects:

$$Q_M = \alpha \frac{RW_{ER}}{RW_S} + (1-\alpha)\frac{RF_{ER}}{RF_S} \tag{14}$$

Since the numerator of Q_M is in terms of effort and its denominator is in terms of SLOC, it is a ratio of productivities (i.e., effort per SLOC). Some simple mathematical rearrangement will show that Q_M is equivalent to:

$$Q_M = \frac{Productivity_{Maintenance}}{Productivity_{Developement}} \tag{15}$$

- **Maintainability Index (MI)** [11][12]: It is based on the product complexity and it is obtained from the following parameters:
  - **aveV**: average Halstead Volume V per module;
  - **aveV (G)**: average Cyclomatic Complexity per module;
  - **aveLOC:** LLOC average per module;
  - **perCM:** average percentage of comments per module;

$$\begin{aligned} MI = [&171 - 5.2 * \log_2(aveV) - 0.23 * aveG - 16.2 * \log_2(aveLOC) \\ &+ 50 * \sin(\sqrt{2.4 * perCM})] \end{aligned} \tag{16}$$

- **Defect density (D_D)**: represents the defects density within the source code:

$$D_D = \alpha \cdot RW_R + (1-\alpha) \cdot RF_R \tag{17}$$

Refactoring. Since the refactoring does not alter the external behaviour of the software, monitoring the quality of the refactoring phase is not a trivial issue. This metric tracks the time course of the refactoring based on the acceptance tests [9]:

- **Running Tested Features (RTF)** [13]: the metric is defined as the number of features that passed their acceptance test.

Agile methodologies do not allow the application to be designed too much in advance, planning the use of a work-in-progress refactoring. Bad refactoring might damage some application functionalities, invalidating the related acceptance tests. In order to plot a growing straight RTF line over time, an Agile team has to follow good refactoring practices along with the use of an automatic acceptance testing framework. So, at the same time, the RTF metric monitors the quality of the refactoring practice and promotes the use of both these Agile techniques.

Continuous Integration. In this section, we describe a simple metric that allows the detection of the Continuous Integration degree implemented in a project:

- **Pulse** [3]: the metric counts the number of commits towards the versioning repository.

As with the RTF metric, the Pulse metric combines monitoring with the promotion of a particular Agile technique.

Process Metrics. Process metrics are useful in monitoring the sequence of activities invoked to produce the software product [14].

- **Progress:** it shows the progress of the development as the number of made tasks.

Resource Metrics. Resource metrics aim to monitor people, methods and tools time, effort and budget [14]:

- **Effort:** it measures the effort required and planned for each staff member (in terms of time, story points, resources, etc.) compared with that actually provided;
- **Team satisfaction:** it is a survey that provides information about the perception of the Agile transformation from the point of view of the staff involved. Examples of questions can be retrieved from [15].
- **Customer satisfaction:** in the Agile methodologies, customer role is essential to provide a valid software product. So that customer can be considered as a resource of the software life cycle. A questionnaire can be useful for collecting customer considerations about team and software product quality.

Project Metrics. Project metrics offer a set of measurements useful in contributing to project control, risk mitigation and to managing team performance [14]:

- **Cost:** cumulative interpretation of the spent effort;
- **Burn-down** [3]: is the opposite of the Cost metric, since it indicates the amount of work still to be done to complete the project (or sprint).

4 Feedback from the Companies

As specified in the previous sections of this paper, the selected metrics were subject to the reviews of the business partners in order to refine the set of metrics and to adapt it to the enterprises' needs. The feedback from the companies was based on both meetings and workshops and on a survey about the companies' viewpoints regarding the proposed metrics. The survey was composed with the questions shown in Table 1. Fig. 1 shows three charts representing the results of the survey. Each chart shows the averaged values of the points given by companies to the questions concerning their opinions about each metric. As illustrated in the first chart, the comprehension of the metrics was generally high. However, the second chart shows that there were some difficulties in the willingness to retrieve them. As it can be gathered from the third chart, this problem had to be related above all to the lack of adequate data collection tools for the enterprises. The issue was also confirmed in the workshops with the partners and in the comments attached to the survey feedbacks.

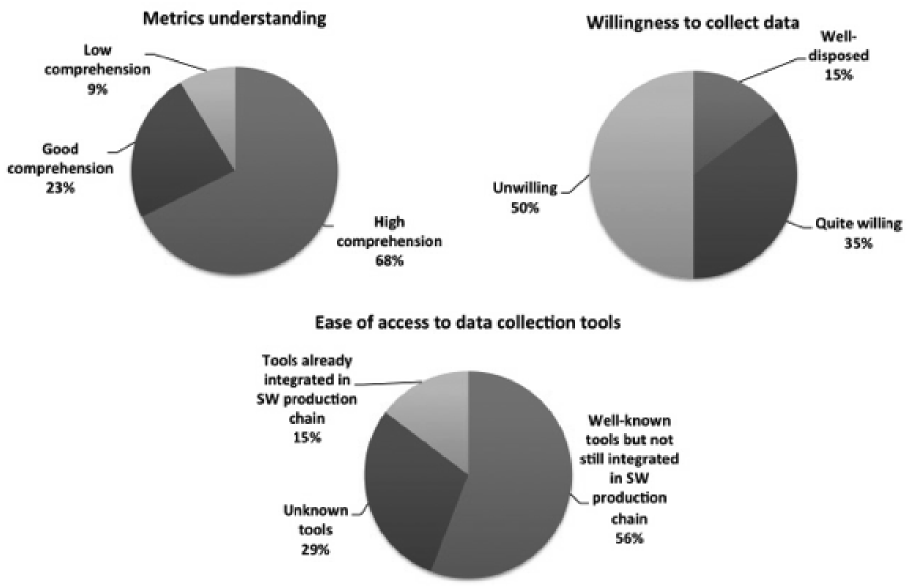

Fig. 1. Survey Result

Therefore, at the suggestion of the companies, the problem was overcome by identifying a set of tools and strategies in support of the collection of the measurements. Through a synergistic exchange of ideas between the companies and the research groups, the metrics set was split into three macro-groups, depending on the different methods and tools used in collecting data. The groups do not cover all the metrics set, because the expressions of some of them are derived and then estimable not directly in the monitoring phase.

The first group concerns the metrics whose measurements have to be obtained through automatic calculation tools, due to their computational complexity (Table 3). The only constraint on the tools choice is of allowing the complete export of the collected data, for later analysis. An example of an automatic calculation tool selected by the companies is Eclipse Metrics, a Java plugin for Eclipse. In the .NET environment, a recommend tool is Ndepend. However, the companies adopted various tools concurrently, each one providing different categories of metrics.

The second group comprises the metrics related to the measurements currently not supported by any tool, but achievable by manual procedures. Table 4 shows the metrics belonging to this group. To support the collection of data, spreadsheet templates have been set up for each one of these metrics, as a sort of wizard.

The third and last group affects those metrics achievable by handling data provided by project management and ATDD tools. These metrics are shown in Table 5.

The ATDD tools provide the information needed for the RTF metric. Storing this simple information in a spreadsheet allows an historical vision of the feature test results. Project management tools, adequately used, represent the data source for quality, process, resource and project metrics. They permit to specify for each task the involved teammate, the estimated and effective effort spent and, in the case of SCOs,

Table 3. Metrics achievable through automatic calculation tools

Metrics category	Metrics
Product Size	Logical Line Of Code Number of classes Number of abstract classes/interface Abstractness
Complexity	Cyclomatic Complexity Halstead Complexity
Object Oriented	Afferent Couplings Efferent Couplings Coupling Between Objects Instability Lack Of Cohesion of Methods Number of Overridden Methods Depth of Inheritance Tree Number of Children Specialization Index Weighted Methods per Class Response for a Class
Code Coverage	Statement Branch/decision Boolean/Condition Loop Path
Quality	Maintainability Index

Table 4. Metrics currently achievable through non-automated procedures

Metrics category	Metrics
Product Size	Test Case Points analysis
Usability	MiLE+
Continuous Integration	Pulse
Resource	Team satisfaction Customer satisfaction

the defect type and the involved lines of code. Some of the companies have used Redmine, a project management web application, extended with Backlogs plugin.

A clever answer to SMART requests has been Police, a flexible SVN plugin developed by one of the companies involved in the project, I.T.S. S.r.l. (Informatica, Tecnologie e Servizi S.r.l.). The plugin allows the definition of custom rules to be applied to the code before each commit towards the SVN repository. The company chose this solution to synchronize agile tasks with its internal project management tool, not currently suitable for agile paradigm.

Table 5. Metrics achievable from project management and ATDD tools

Metrics category	Metrics
Refactoring	Running Tested Features
Quality	Critical Defects (SCO_1) Improvement Requests (SCO_2) Normal Defects (SCO_3) Open Reworks Closed Reworks Rework Effort Open Refactors Closed Refactors Refactor Effort
Process	Progress
Resource	Effort
Project	Cost Burn-down

5 Related Work

The selection of the metrics is based on a review of the already completed work on the monitored migration from classical development methodologies to Agile practices. Below we describe some of them, highlighting the key points that have been useful for our work. Furthermore, in the data analysis the measurements related to the metrics will assume the form of a complex data source, composed of a heterogeneous set of information repositories. So, data mining processes should be applied to the latter in order to obtain consistent information about the trend of experimental projects. For this reason, in this section, we provide a brief introduction to the state of the art of the practices of mining software repositories.

A survey of the Agile metrics and their comparison with traditional development metrics is provided in [2]. Several metrics are listed according to the different categories they belong to, claiming that both methodologies utilize basically the same sets of metrics and classifications. Simply, some differences lie in the use of metrics more closely describing teamwork and Agile aspects, such as team effort, velocity, story points, and so on.

An innovative project, described in [5], has implemented Agile processes and XP practices in a software development team belonging to the Israeli Air Force. As the army is a large and hard-to-change organization with a rigid organizational and managerial structure, the transition to Agile methods was challenging. Thus, such a change was performed in a gradual carefully planned process and it was adapted to the team involved in the study, comprising 60 skilled developers and testers with different individual interests. The project affected by the transition was large-scale, enterprise-critical software, in which quality and fit to customers' needs were not to be compromised by the study. To ensure this and to communicate information on the project's trend to the team, a set of four metrics was selected. They indicated the amount and the quality of work performed and the work status and progress. The four metrics discussed are: (i) *Product Size*, presenting the amount of completed work based on tests written for the application; (ii) *Pulse*, counting how many check-in operations occur per day; (iii) *Burn-down*, providing information about the remaining project work versus the remaining human resources; and (iv) *Faults*, giving the faults per iteration. The use of the aforementioned metrics mechanism increased the confidence of the team members as well of the unit's management with respect to using agile methods.

In [16], Waterfall and Extreme Programming techniques are empirically compared with the aim of looking at the advantages and disadvantages of the methods. The experimental period was five years and the results of the study are based upon the outcomes, generated artefacts and metrics produced in reality by different teams involved in the same project. Some of them used Waterfall development methodologies and the others Agile methodologies. To learn about the effective transition from traditional to agile development methods, the study was conducted at Carnegie Mellon University in Silicon Valley. The metrics used to compare the two methodologies have been divided into three categories: (i) *requirements metrics*; (ii) *design metrics*; and (iii) *implementation metrics*.

First of all, the study points out the fact that setting up this kind of experiment so much in advance is challenging. The paper shows that Waterfall teams spent more time creating high formal documents whereas Extreme Programming teams spent more time writing code and documenting the design and the code. Also, the amount of code and features completed were roughly the same for both methods.

The study described in [15] tries to provide evidence of the impacts of Agile adoption in a very large software development environment, Nokia, from the point of view of the teams involved in the development. A population of more than 1,000 respondents in seven different countries in Europe, North America, and Asia was subjected to a questionnaire. The results reveal that most respondents agree on all accounts with the generally claimed benefits of agile methods. These benefits include higher satisfaction, a feeling of effectiveness, increased quality and transparency, increased autonomy and happiness, and earlier detection of defects. Also, 60% of respondents would not like to return to the old way of working. Beyond this further evidence of the benefits introduced by Agile methods, the study provides an innovative kind of test for evaluating a software development project.

Hereafter we show some examples about the mining software repository practice. It will be essential in order to analyse the measurements retrieved by the proposed metrics framework. In the literature, various methodological frameworks there exist for the application of this practice to software engineering, for example [17].

Mining software repositories has several goals and can lead to different results. For example, in [18], the authors implemented a static source code checker to drive and to help refine the search for bugs. The goals of the study rested upon the data retrieved from the source code repository. In order to refine its results, the system searched for a commonly fixed bug and used information automatically mined from the repository.

The application of this technique on real projects, like Apache web server and Wine, shows that it is more effective than the common static analysis.

The authors of [19] present a framework with the purpose of combining different software repositories in order to ease mining process applications. The framework is called FRASR (FRamework for Analysing Software Repositories). It applies systematic data mining pre-processing procedures, combining data from different sources and extracting logs useful for process mining applications. The study involved various data sources, like versioning repositories, task managers, mailbox archives, and so forth.

Our study aims to differentiate itself from the related work by two main key points: the richness of the software metrics set and the tightly collaborative approach instantiated with the IT business world.

6 Conclusion and Future Work

The presented study is part of the SMART project, which aims to enable a systematic approach with real software factories in order to help them in their transition from Waterfall to Agile methodologies. With this paper, we have described a complete set of metrics useful in monitoring different aspects of the software life cycle. We did not propose new metrics, but we have selected the more suitable ones to monitor software projects developed with Agile methods. Empirical data provided by enterprises involved in the project allowed us to select only the more interesting and applicable metrics from the businesses' points of view, discarding those too difficult to integrate into the software development processes. This software metrics framework will be applied to several experimental projects in transition towards Agile methods in the forthcoming period, so monitoring their trends.

The study started from the analysis of the research context. First of all, we examined the experimental software project and the involved teams to give a complete perspective of the technological environment. This analysis revealed the strong heterogeneity of the technologies used and the lack of any kind of Agile practices in the projects.

Subsequently, we presented a large set of metrics, suitable to monitor each aspect of the software development phases, fitting it to the different technological conditions imposed by the heterogeneous projects. The metrics have been selected to monitor the migration towards Agile development methodologies, promoting the practices strictly related to them. The set of metrics was subjected to the project partners review in different meetings and by a survey. We analysed the feedback in the paper, focusing on the difficulties in retrieving the measurements by the enterprises due to a lack of known retrieval tools and strategies. The partners review revealed their own importance also in selecting some of the presented metrics.

Subsequent work will be related to the analysis of the measurements retrieved, applying the presented plan of measures to the experimental projects. The time traces of the collected data will show the effective impact of Agile methods in relation to software projects already started with classical development methods. Therefore, future studies will aim to show the benefits and disadvantages of Agile, introducing scientific innovation because of the numerous monitored aspects and the specific kind of environment under study.

References

1. Beck, K., Beedle, M., Bennekum, A., Van, C.A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for Agile Software Development, `http://agilemanifesto.org/`
2. Misra, S., Omorodion, M.: Survey on agile metrics and their inter-relationship with other traditional development metrics. ACM SIGSOFT Softw. Eng. Notes 36, 1 (2011)
3. Nguyen, V., Pham, V., Lam, V.: qEstimation: a process for estimating size and effort of software testing. In: Proceedings of the 2013 International Conference on Software and System Process, ICSSP 2013, p. 20. ACM Press, New York (2013)
4. Nguyen, V., Pham, V., Lam, V.: Test Case Point Analysis: An Approach to Estimating Software Testing Size, `http://www-scf.usc.edu`
5. Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall PTR (2003)
6. Dubinsky, Y., Talby, D., Hazzan, O., Keren, A.: Agile metrics at the Israeli Air Force. In: Agile Development Conference (ADC 2005), pp. 12–19. IEEE Comput. Soc. (2005)
7. Triacca, L., Bolchini, D., Botturi, L., Inversini, A.: MiLE: Systematic usability evaluation for e-learning web applications. In: Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications, Chesapeake, VA, pp. 4398–4405 (2004)
8. Zhu, H., Hall, P.A.V., May, J.H.R.: Software unit test coverage and adequacy. ACM Comput. Surv. 29, 366–427 (1997)
9. Kunz, M., Dumke, R.R., Zenker, N.: Software Metrics for Agile Software Development, pp. 673–678 (2008)
10. Royce, W.: Pragmatic Quality Metrics for Evolutionary Software, TRW Space and Defence Sector, Redondo Beach, California (1990)
11. Khan, R.A., Mustafa, K., Ahson, S.I.: Software Quality: Concepts and Practices (2006)
12. Ganpati, A., Kalia, A., Singh, H.: A Comparative Study of Maintainability Index of Open Source Software. Int. J. Emerg. Technol. Adv. Eng. 2, 228–230 (2012)
13. A Metric Leading to Agility, `http://xprogramming.com/articles/jatrtsmetric/`
14. Royce, W.: Software Project Management: A Unified Framework (1998)
15. Laanti, M., Salo, O., Abrahamsson, P.: Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. Inf. Softw. Technol. 53, 276–290 (2011)
16. Ji, F., Sedano, T.: Comparing extreme programming and Waterfall project results. In: 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE & T), pp. 482–486. IEEE (2011)
17. Hassan, A.E., Xie, T.: Mining software engineering data. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ICSE 2010, p. 503. ACM Press, New York (2010)
18. Williams, C., Hollingsworth, J.: Automatic mining of source code repositories to improve bug finding techniques. IEEE Trans. Softw. Eng. 31, 466–480 (2005)
19. Poncin, W., Serebrenik, A., Van Den Brand, M.: Process Mining Software Repositories. In: 15th European Conference on Software Maintenance and Reengineering, pp. 5–14. IEEE (2011)