Combining Extreme Programming with ISO 9000 *

Jerzy R. Nawrocki,
Michał Jasiński, Bartosz Walter, and Adam Wojciechowski

Poznan University of Technology, ul. Piotrowo 3A, 60-965 Poznan, Poland
{Jerzy.Nawrocki, Michal.Jasinski, Bartosz.Walter, Adam.Wojciechowski}
@cs.put.poznan.pl
http://www.cs.put.poznan.pl

Abstract. The main drivers of the growing ICT market are software products. European Information Technology Observatory estimates, that in year 2002 the total value of ICT software products in Western Europe will be more than 70 billions Euro. Unfortunately very few people are satisfied with quality of the software products and processes. Software Process Improvement tools, like CMM and ISO 9000 were to cure this situation, but some people complain that they are too bureaucratic and inflexible. As a result new, so-called *agile*, methodologies appeared. One of them is Extreme Programming (XP) - a lightweight, change-oriented and customer-oriented approach to software development. Although XP proposes many interesting practices, it has some limitations. Moreover, it is not clear how to introduce XP to an organization certified to ISO 9001:2000. The aim of the paper is to present a modified version of XP that would be acceptable from the point of view of ISO 9000.

1 Introduction

European Information Technology Observatory (EITO) predicts that the total value of Western Europe ICT market will reach 678 billion Euro in 2002 [7] and 11% of it (more than 70 billions Euro) will be spent on software products. However, the market is getting more and more demanding. To be successful, software companies have to attract customers in various ways. One of possible steps is certification to ISO 9001:2000.

ISO 9000 is a series of international standards concerning establishment and maintenance of a quality management system. They are general-purpose standards comprising vocabulary [3] requirements [4] and recommendations for improvement [5]. The only standard an organization can be certified to is ISO 9001. They can be used in a private factory as well as in a government institution, in a big shipyard and in a small software company. ISO 9000 originated in UK and it is getting more and more popular. In 1997 there have been approximately 102 000 registrations worldwide and three years later the number was over 250 000 [15].

* This work has been financially supported by the State Committee for Scientific Research as a research grant 4 T11F 001 23 (years 2002-2005)

Opinions on ISO 9000 vary significantly. For instance, the Director General of the British Standard Institute claimed that ISO 9000 *"will save your money"*, *"it will ensure satisfied customers"*, and *"it will reduce waste and time-consuming reworking of designs and procedures"* [15]. At the opposite pole is the opinion of John Seddon. According to him *"by being labelled a quality standard, ISO 9000 has only succeeded in steering quality into troubled waters. Far from being a first step to quality it has been a step in the wrong direction. The hope is that it hasn't conditioned management to lose interest in the subject"* [15]. That opinion is somehow confirmed by the fact, that almost 10% of Australian companies have decided to discontinue registration to ISO 9001 [15]. What is wrong with ISO 9000? The general impression is that ISO 9000 standard requires too much documentation and it is too bureaucratic. Moreover, some people consider ISO 9000 too general and different types of software-oriented maturity models have been proposed [16, 14, 6]. Nevertheless, some software companies decided to go through the ISO 9001 certification process. The danger is that certification to ISO 9000 will result in a well-documented but still inefficient system. The initial enthusiasm of workers for the software process improvement will soon be dissipated if they find out that the ISO 9001 is just a marketing subterfuge, and the company has no intention to introduce a real process improvement.

On the other hand, a few years ago so-called *lightweight* (or *agile*) software development methodologies have appeared. The most popular is Extreme Programming (*XP* for short). XP emphasizes the importance of on-site customer, oral communication, product quality, short feedback from customer and end-users, simplicity, minimal documentation and avoidance of overtime. In the context of Information and Communication Technology it is important that XP is a change-oriented methodology and it tries to deliver maximum functionality at a minimum cost in a short time. A typical reaction of a programmer to XP is: *"At last a methodology for people, not people for methodology"*. Our idea is to use this enthusiasm as a starting point for real software process improvement and to combine it with the requirements of ISO 9000. Unfortunately, that merge is not straightforward. What we propose is a modified methodology, based on XP practices and conformant to ISO 9001:2000.

In the next section we will present most important features of Extreme Programming. Then, in section 3, we will discuss the Quality Management System of an ISO-9000 software company. We will show how to put together a general Quality Management System proposed by ISO 9000 and a software-oriented knowledge base which contains documents, artifacts, and data specific to a software organization. Our focus will be on two parts of ISO 9001:2000: product realization, and monitoring and measurement. Product realization, which is based on XP practices and conformant to ISO 9001:2000, will be described in section 4. To gain flexibility, we propose a four-level improvement schema resembling the Capability Maturity Model [14]. In section 5 we will describe measurements which are necessary from the point of view of ISO 9000 and useful in the context of XP practices. In the last section our early experience concerning the proposed approach will be presented.

2 XP Overview

Extreme Programming [1, 2, 8] represents a new wave in software development known as the *approach*. Tom de Marco, the father of structural analysis, calls XP the most important movement in software engineering (see the foreword to [2]). The strong points of XP in the ICT context are as follows:

- *Risk minimization*. ICT is developing very fast. To catch up with current developments it is necessary to make investments in new technologies and try new tools out. On the other hand, new tools and technologies are immature and one cannot depend on them. The best approach is to make some (preferably small) investment now and after some time invest more or give up, depending on the developments (it is like buying an option on the stock exchange). XP is based on incremental software development and its suites the strategy very well.
- *Customer orientation*. In XP all the business decisions are made by the customer and he has the full control over the development process.
- *Lack of excessive paperwork*. In XP programmers concentrate on programming, not on writing documentation. The only artifacts they have to produce are test cases and code.
- *Quality assurance through intensive testing*. In XP programmers first create test cases then they write code. Automated tests and integration are performed several times a day and they drive the development process.
- *Lack of overtime*. Short releases and increments allow to gain experience very fast. This makes planning easier and more dependable. As a result programmer do not have to (always) work overtime.

XP has also weak points. The most important are problems with software maintenance. Since the only artifacts are test cases and code, after some time it can be very difficult to maintain the software. It would be also the problem from the ISO 9000 point of view. In the remaining part of the paper we propose how to solve that problem.

3 Software Development in an ISO 9000 company

ISO 9001:2000 standard defines requirements for a process-oriented Quality Management System (QMS for short). This means that desired results are achieved more efficiently when the related resources and activities, together with encompassing customer needs and satisfaction, are managed as a process. QMS is specified in a Quality Manual document featuring a three-tier structure, which consists of Quality Processes (including Quality Policies), Quality Procedures and Work Instructions. This structure is presented in Fig. 1.

The problem is that Work Instructions are sometimes too bureaucratic. A good example of that approach is Tricker's book on ISO 9000 [18]. According to it, a Work Instruction takes about 16 pages. Half of them contains purely administrative data (document data sheet, distribution list, amendments, list of

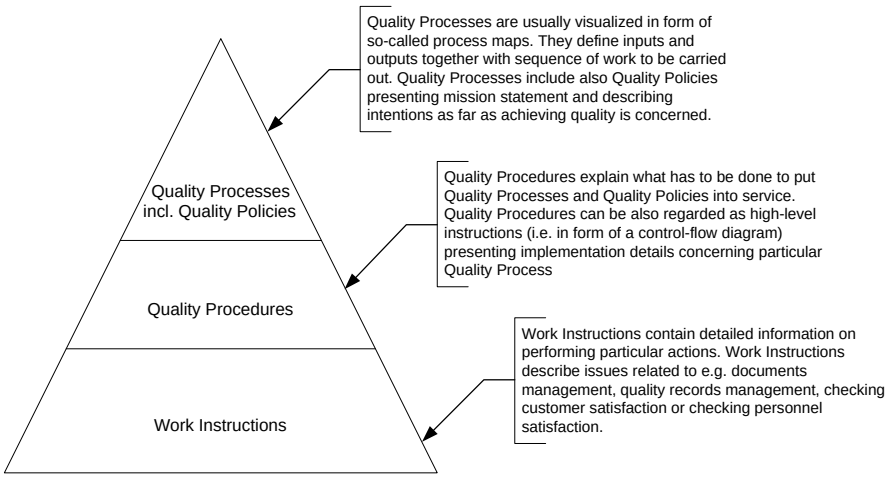

Fig. 1. Three-tier structure of Quality Manual.

annexes etc.). That makes the whole QMS documentation superfluously thick. Another drawback of Tricker's approach is form-orientation: Work Instructions focus on how to fill-in the forms used by the Quality Procedures. What we propose is to make Work Instructions shorter (some elements can be omitted, some, e.g. terminology, can be put together and placed in one section). Moreover, Work Instructions should describe practices specific for a given methodology of software development.

In our opinion, quality organization needs two things: general Quality Management System operating on a high abstraction level and a *Thesaurus* (knowledge database), which should materialize company's knowledge. In the thesaurus templates of e.g. Quality Plans, historical data concerning past projects etc can be deposited. This information will be indispensable during planning and improving software processes.

The clauses of ISO 9000:2000 can be split into two parts. One part describes the general Quality Management System (chapters 4, 5, and 6) while the other part specifies requirements for a methodology to be adopted by an ISO-9000 company (chapters 7 and 8 of ISO 9001:2000). In the remaining part of the paper we will focus on requirements imposed by chapters 7 and 8 of the ISO 9001:2000.

4 Product Realization: Modified XP Practices

To adopt XP practices to the needs of ISO-based product realization we suggest to use the XP Maturity Model (XPMM for short)[12]. That model resembles the SEI's Capability Maturity Model[14]. The XPMM's Key Process Areas associated with the maturity levels include: Customer Relationship Management and

Product Quality Assurance (Level 2), Pair Programming (Level 3), and Project performance (Level 4).

The XPMM maturity levels are useful as they allow to introduce XP to a company gradually (level by level). That provides flexibility to the software improvement process.

In the next section we describe XPMM's Key Process Areas together with modification, which are necessary to make the software development process compliant with ISO 9000:2000.

4.1 Customer Relationship Management

Customer satisfaction is of primary importance for both XP and ISO 9000. However, XP is much more specific about how to obtain that satisfaction. Here are the main XP practices directly influencing customer satisfaction:

CRM1: *User stories are used to describe requirements.* They are written on small pieces of paper using natural language. Usually the are very short and are just an introduction to the discussion between customer and the development team, so they do not have to be complete. User stories are not documented and maintained.

CRM2: *A development process is split into short releases (about 6-9 weeks) and each release is split into iterations (2-3 weeks).* Each iteration implements a set of user stories. When a release is finished the product is made available to the end users. This provides a fast feedback to the development team.

CRM3: *Planning game is used to create a release plan.* The user stories brought by the customer are evaluated by the developers. They estimate the effort required to implement each story and the technical risk. Knowing that, customer chooses the stories to be implemented in the current release.

CRM4: *A metaphor is chosen to facilitated communication with the customer.* The system is described in terms the customer is familiar with.

CRM5: *No functionality is added early.* The functionality to be implemented must be chosen by a customer representative, not by the development team.

The first problem, in the context of ISO 9000 is lack of written requirements (XP uses user stories instead of documented requirements). ISO 9001:2000 does not explicitly specify the need for existence of written requirements. However, the need for existence of written documentation follows from these clauses:

- *"(. . .)the organization shall determine (. . .) records needed to provide evidence that the (. . .) resulting product meets requirements."*([4], clause 7.1),
- *"The organization shall review the requirements related to the product (. . .) where the customer provides no documented statement requirement, the customer requirements shall be confirmed by the organization before acceptance."* ([4], clause 7.2.2),
- *"Inputs relating to product requirements shall be determined and records maintained. These inputs shall include functional and performance requirements (. . .). Requirements shall be complete, unambiguous and not in conflict with each other."*([4], clause 7.3.2).

The need for documented requirements is also emphasized by the CMM model ([14], KPA for Requirements Management, Ability 2). Thus, the problem arises how to introduce documented requirements to a lightweight methodology. To solve it one has to notice that XP was created to be lightweight to programmers and it puts extra work on the shoulders of other people, e.g. customer representative. We will follow that path. One of the roles in an XP team is the one of a tester, who implements test cases proposed by the customer. Our suggestion is to make the tester responsible for requirements documentation and management.

According to Beck, an XP tester is *"responsible for helping the customer choose and write functional tests"* and running them regularly [1]. Requirements and acceptance tests are on the same abstraction level, so tester seems to be the best person to take on the job of analyst responsible for requirements management.

4.2 Product Quality Assurance

Product quality assurance is addressed in ISO 9000 by clause 7.3.5 *Design and verification*: *"Verification shall be performed in accordance with planned agreements to ensure that the design and development outputs have met the design and development input requirements."*

XP implements this through the following practices:

PQA1: *Test-first coding.* It means that a programmer first writes a test, then starts coding. This helps to understand what we expect from a unit and removes "implementation bias" during testing.
PQA2: *All code must have unit tests.* That allows regression testing.
PQA3: *When a bug is found a test must be created.* That supports regression testing.

Moreover, XP supplements those practices with two others:

PQA4: *Continuous integration.* This should provide fast feedback on current system status.
PQA5: *Optimization is left till last.* Many optimizations require lots of effort and they are sources of potential bugs which are difficult to locate and fix. Thus, it is better for the product quality not to introduce and optimization if it is not necessary.

4.3 Pair Programming

Pair programming is specific to XP and it does not relate directly to any ISO 9000 clause. However, to implement pair programming in an efficient way, an open workspace lab is required. That is connected with clause 6.3 *Infrastructure* and 6.4 *Work environment*. Moreover, it is interesting to point out that ISO 9000 requires *conformity to product requirements*, not to *the defined processes*.

The pair programming practices are as follows:

PP1: *Code must be written to agreed standards.* This way it is much easier to understand and modify the code written by somebody else.
PP2: *All production code is pair programmed.* That is the main practice for XP projects.
PP3: *Only one pair integrates code at a time.* It is needed to ensure code consistency.
PP4: *Collective code ownership.* Everybody can change any piece of code if necessary.
PP5: *Use version management system.* It supports collective code ownership and continuous integration.

5 Monitoring and Measurement

In ISO 9001:2000 monitoring and measurement are described in very general terms. Extreme Programming gives hints how to implement ISO 9001 clauses related to this issue:

- *8.2.3 Monitoring and measurement of processes.* The aim is to *"demonstrate the ability of processes to achieve planned results"*. In the context of XP one should collect the following process metrics:
  - *Overtime* (day by day). XP assumes no overtime and one should know how far we are from the ideal process.
  - *Availability* of the customer representative to the development team to answer questions, resolve conflicts and create acceptance tests
  - *Project velocity* (i.e. the amount of time per week each team member can spend on his assignments). That data are used during planning.
  - *Integration log* to see how frequently new pieces of code are integrated with the system (XP suggests to have several integrations per day).
  - *Mode of production* for each piece of production code (pair or individual).
  - *Programming speed* (lines of code per hour, test cases per hour, acceptance tests per hour).
- *8.2.4 Monitoring and measurement of product.* In the context of software development the main measurement is test report showing the fraction of passed unit tests and acceptance tests (both unit tests and acceptance tests can comprise functional and performance tests).
- *8.3 Control of nonconforming product.* ISO 9001:2000 requires that *"when nonconforming product is concerned it shall be subject to re-verification to demonstrate conformity to the request"*. In XP before a new version of the system is checked-in to the baseline library all the unit tests should be passed.
- *8.5.2 Corrective action.* According to ISO 9001:2000 it is necessary to *"eliminate the cause of nonconformities in order to prevent recurrence."* To prevent recurrence of software defects XP requires to create test cases for each detected defect. Should the defect recur, the test cases will discover it before a new version of the system will be released.

The remaining clauses concerning monitoring and measurement (customer satisfaction, internal audit, analysis of data, continual improvement and preventive action are transparent with regard to XP.

6 Conclusions

The approach described in the paper is being implemented at the Software Development Studio (SDS for short). SDS is a software organization established at the Poznan University of Technology to allow students to get practical experience in software development and software process improvement [9]. Each year there are 11 projects developed for real customers. Each project involves 8 students: 4 from 3rd year, 2 from 4th year and 2 from 5th year of studies. Students play different roles in subsequent years, gaining experience in design and programming, project management and quality assurance. Each project lasts an academic year (9 months). The examples of the projects developed by students include: *Network Database System for Multiple-Choice Questions*, *Internet-Based Environment for Requirements Management*, *Internet-Based Traffic Analysis System.*

In the previous academic year the projects were split into two groups. One group developed software according to CMM Level 2, and the other applied pure XP. The XP projects suffered, among others, from: absence of precisely defined process, lack of adequate communication with customer, late delivery, and the most significant - maintenance problems. One of symptoms of that problems were the difficulties the 3rd year students had in writing their bachelor thesis [13].

This year we have used a modification of XP targeting at satisfying selected ISO 9000 clauses embracing requirements management and maintenance problems [10]. The aim of this experiment was to introduce ISO 9000 elements to XP approach.

In order to asses maturity of requirements engineering processes in SDS projects we used the Somerville-Sawyer model [17] based on a set of *good practices*. Sommerville and Sawyer have identified 66 practices and split them into 3 groups: basic, intermediate and advanced. Each practice can bring from 0 to 3 points, depending on how widely it is used by an organization. The highest maturity level is called *Defined*. Organizations at that level have more than 85 points in the basic practices and more than 40 points in the intermediate and advanced practices. The intermediate level is called *Repeatable* and organizations at this level have more than 55 points in basic practices. The lowest level is called *Initial* and an initial organization has fewer than 55 points in basic practices.

Our experiment shows that Extreme Programming modifications resulted in significant improvement from the Somerville-Sawyer point of view. Since XP addresses only 6 of the basic practices, 4 intermediate and 1 advanced practice, *"classical" XP* projects were assessed as Initial [10]. In contrast, this year projects developed according to the proposed methodology supported most of the 66 practices more or less directly and therefore were assessed as Repeatable. That can be considered as an important improvement indicator. However there is still a need for further research. These studies shall focus on implementing a Quality Management System combining ISO 9000 and XP.

References

1. Beck, K.: Extreme Programming: Embrace Change. Addison-Wesley, Boston (2000)
2. Beck, K., Fowler, M.: Planning Extreme Programming. Addison-Wesley, Boston (2001)
3. European Committee for Standardization: Quality Management Systems - Fundamentals and Vocabulary (ISO 9000:2000). European Committee for Standardization (2000)
4. European Committee for Standardization: Quality Management Systems - Requirements (ISO 9001:2000). European Committee for Standardization (2000)
5. European Committee for Standardization: Quality Management Systems - Guidelines for Performance Improvements (ISO 9004:2000). European Committee for Standardization (2000)
6. European Committee for Standardization: Software Process Assessment (ISO 15504:1998). European Committee for Standardization (1998)
7. European Information Technology Observatory: EITO2002 - 10th Edition 2002. European Information Technology Observatory, Brussels (2002)
8. Jeffries, R., Anderson, A., Hendrickson., C.: Extreme Programming Installed. Addison-Wesley, Boston (2001)
9. Nawrocki, J.: Towards Educating Leaders of Software Teams: A New Software Engineering Programme at PUT. Proceedring of SEES 98. Scientific Publishers OWN, Poznan (1998) 149–157
10. Nawrocki, J., Jasiński, M., Walter, B., Wojciechowski, A.: Extreme Programming Modified: Embrace Requirement Engineering Practices. Proceedings of the 10th IEEE Joint International Requirements Engineering Conference. IEEE Press, Inc., Los Alamitos (2002) 303–310
11. Nawrocki, J., Wojciechowski, A.: Experimental Evaluation of Pair Programming. In: Maxwell, K., Oligny, S., Kusters, R., van Veenendaal, E. (eds.): Project Control: Satisfying the Customer. Proceedings of ESCOM 2001. Shaker Publishing (2001) 269–276
12. Nawrocki, J., Walter, B., Wojciechowski, A.: Toward Maturity Model for eXtreme Programming. Proceedings of the 27th EUROMICRO Conference, Los Alamitos. IEEE Computer Society (2001) 233–239
13. Nawrocki, J., Walter, B., Wojciechowski, A.: Comparison of CMM Level 2 and eXtreme Programming. Proceedings of the 7th European Conference on Software Quality, Helsinki, Finland. Lecture Notes in Computer Science 2349, Springer-Verlag (2002) 288–297
14. Paulk, M. C. et al.: The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, Reading MA (1994)
15. Seddon, J.: The Case Against ISO 9000. 2nd edn. Oak Tree Press, Dublin (2000)
16. Software Engineering Institute: Capability Maturity Model Integration. Version 1.1, Staged Representation. Carnegie Mellon University (2002)
17. Sommerville, I., Sawyer, P.: Requirements Engineering: A Good Practice Guide. John Wiley & Sons, Chichester (1997)
18. Tricker, R., Sherring-Lucas, B.: ISO 9000:2000 in Brief. Butterworth-Heinemann, Oxford (2001)