Available online at www.sciencedirect.com
SCIENCE DIRECT
Information and Software Technology 47 (2005) 257–269
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Software development productivity on a new platform: an industrial case study

Piotr Tomaszewski*, Lars Lundberg

School of Engineering, Blekinge Institute of Technology, P.O. Box 520, S-372 25 Ronneby, Sweden

Received 3 March 2004; revised 22 August 2004; accepted 23 August 2004
Available online 2 October 2004

Abstract

The high non-functional requirements on mobile telecommunication applications call for new solutions. An example of such a solution can be a software platform that provides high performance and availability. The introduction of such a platform may, however, affect the development productivity. In this study, we present experiences from research carried out at Ericsson. The purpose of the research was productivity improvement and assessment when using the new platform. In this study, we quantify and evaluate the current productivity level by comparing it with UNIX development. The comparison is based on two large, commercially, available systems. We reveal a factor of four differences in productivity. Later, we decompose the problem into two issues: code writing speed and average amount of code necessary to deliver a certain functionality. We assess the impact of both these issues. We describe the nature of the problem by identifying factors that affect productivity and estimating their importance. To the issues identified we suggest a number of remedies. The main methods used in the study are interviews and historical data research.

Keywords: Development environments; Productivity; Software metrics; Improvement; Technology adoption

1. Introduction

Handling the rapid growth of the number of services and subscribers in telecommunication networks has become a very challenging engineering task. Apart from high performance and high capacity, the systems that create the infrastructure for the mobile telecommunication network must provide high availability. No downtime is accepted since it results in huge losses. Different telecom system developers deal with high availability in different ways. There are hardware-based solutions, which main purpose is to avoid a system crash, as well as software-based solutions, that try to handle the situation after a system crash. As a result of the latter approach, a new server platform was introduced by Ericsson. The platform has features that facilitate development of systems with strong high availability requirements. The first experiences after the change of the platform revealed an increase of development time and cost. Both of them affect time-to-market, which is a crucial factor for economically successful software development.

In this paper, we look at two large industrial projects at Ericsson. These projects concern the development of similar products. One project uses the new server platform and the other one uses a traditional UNIX development environment. By comparing time reports from the two projects and conducting interviews, we were able to assess and compare the productivity for the two environments. A well-known problem when dealing with productivity measures is the lack of metrics for measuring the size of the software. The most commonly used metric is the number of code lines [3,4,21,30]. We will discuss this and alternative metrics later in the paper. We also consider ways to measure the quality and complexity of the code, and not only the size. Furthermore, we identify some productivity bottlenecks; one category of these bottlenecks has to do with lack of experience. It is well known from many application domains that tacit knowledge, that has to be acquired by

* Corresponding author. Tel.: +46 457 385876; fax: +46 457 27125.
E-mail addresses: piotr.tomaszewski@bth.se (P. Tomaszewski), lars.lundberg@bth.se (L. Lundberg).

0950-5849/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2004.08.007

long term use and experience, is one source of initial productivity problems when introducing new technology [10], but as people get more used to the new technology these problems should go away.

This study was planned as one of the activities aiming at understanding and improving of the development productivity in the new environment. The initial analysis resulted in formulation of the following research questions:

- *How large is the productivity problem*? When the study was started there were no hard proofs that productivity has actually decreased when the new platform was introduced. The knowledge about satisfactory productivity level and productivity level on the new platform would allow us to quantify the problem. To achieve that we must measure the productivity of software development on the introduced platform and, to obtain a point of reference, compare it with the productivity level in another project, in which the productivity was perceived as good. As discussed above, two projects were selected for comparison:
  - Project A representing UNIX development. The productivity in that project was perceived as satisfactory. This project resulted in Product A.
  - Project B representing the development on the introduced platform. It resulted in Product B.

  Both systems are large (approximately 40–60 man years, 100–200 KSLOC), commercially available, high quality systems that are part of mobile telephone network.
- *Why does the problem occur*? To solve the problem we must identify the issues that cause it. These issues would indicate areas in which there are opportunities for improvement.
- *What can be done about it*? For the issues identified we suggested the remedies.

This paper is an improved and extended version of a previous conference paper [27]. In the current paper, we present new and additional data and expand the discussion concerning quality aspects and the lessons that can be learned from our case study. We also include a significantly expanded discussion about related work.

2. Presentation of the platform

The server platform introduced by Ericsson is usually used in real-time telecommunication applications. This type of applications is characterized by very strong non-functional requirements, like the need for scalability, high availability and efficiency. On the other hand, market demand for lowering maintenance costs and the best price/performance ratio forces the use of standard hardware components. The new platform meets these requirements. The hardware platform, presented in Fig. 1, comprises:

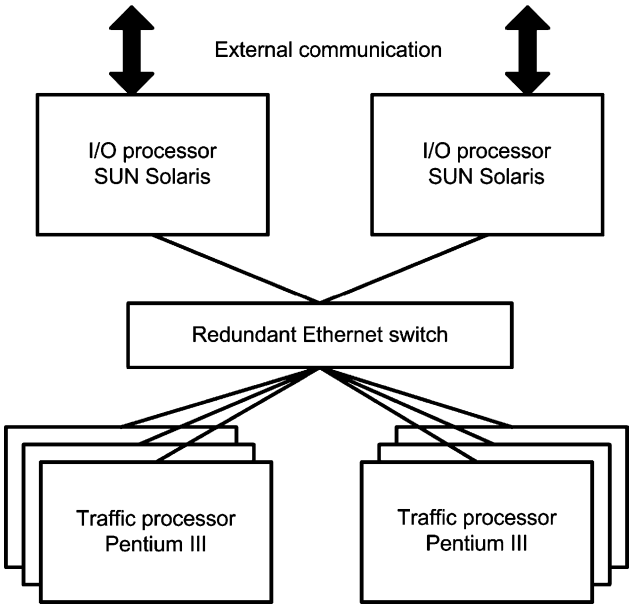

Fig. 1. The hardware configuration of the platform.

- A number of traffic processors that process pay-load. These are Intel Pentium III processors, and each of them has its own memory.
- Four I/O processors responsible for the external communication, maintenance and monitoring of the whole system. These are standard Sun machines running Solaris.
- Two Ethernet switches and two separate interconnections via Ethernet networks.

Although the platform offers standard interfaces (APIs) for Java and C++, the programming model is unique. The main execution unit is a process. There are two types of processes, static ones that are always running and dynamic ones that are created and destroyed on request. The inter-process communication is done by dialogue objects or globally accessible database objects. Dialogue objects are used for message passing communication (Fig. 2). In the communicating processes, two corresponding Dialogue type objects have to be created. They exchange messages using built-in mechanism provided by the platform.

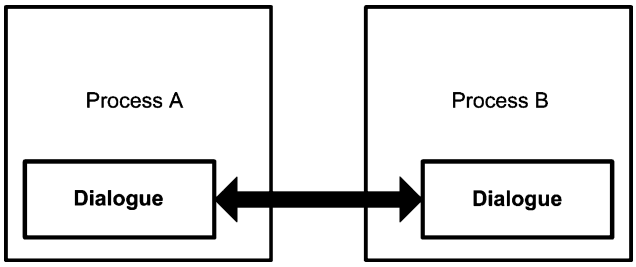

Fig. 2. Inter-process communication using dialogue objects.

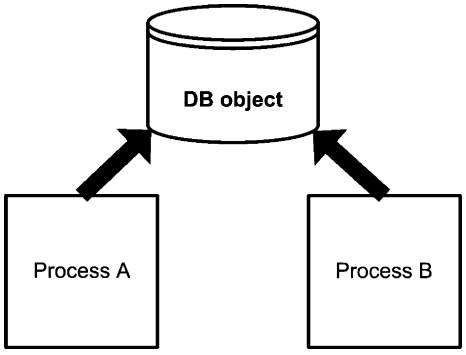

Fig. 3. Inter-process communication using database objects.

Database objects are the basic units of storage in the internal database. They can be accessed by any process running on the platform. They can therefore be used to implement a shared-memory communication model (Fig. 3).

In order to assure an efficient load balance, the programmer has a set of methods for specifying the allocation of database objects and processes to processor pools, i.e. sets of traffic processors on which database objects and processes may end up. The load balancing within a pool is done by the platform itself.

The platform facilitates programming of the highly available systems, i.e. every process or database object is automatically replicated on different machines in the cluster—a crash of one of them does not affect the correct operation of the whole system. Additionally, the platform has built-in features that allow online upgrades of the applications that operate on the platform.

The two applications examined in our study co-operate within the same system that works in the service layer of the mobile telephony network. In each of them the high availability requirement was provided in a different way. In Product A, high availability is assured by a backup server that takes over the work when the main server fails. Product B uses the new platform features to provide high availability.

Both applications have similar design structure. The following subsystems can be identified:

- Platform—software that extends functionalities provided by platform.
- Communication—software responsible for handling of communication protocols.
- Functionalities—software that contains actual business logic of the application.

Both systems are written in C++. To minimize the impact of software reuse on the productivity measurement, only the first versions of both products were taken into account—both were written 'from scratch'. Additionally, it should be noticed that Product B was one of the first projects done on the new platform by the team of developers examined in the study. Before taking part in the project the project members underwent a training program about the new platform. The training program comprised of a one-week long course. Additionally, the developers were provided with a web-based tutorial that covered basic issues connected with programming on the new platform. They also produced a number of prototypes to gain practical, 'hands-on' knowledge about the platform. According to the majority of the developers, the quality of the introduction process was not satisfactory. They suggested that they would benefit significantly from longer and more advanced training.

3. Methods

In this section, we present methods used in the study. Each subsection in this section has a corresponding subsection in Section 4, where the results are presented.

3.1. Productivity measurement

The first thing that must be done is establishing what productivity is and how it can be measured. The traditional productivity definition as a *ratio of output units produced per unit of input effort* [1] is not easily applicable to software development. The input effort is usually defined as the sum of all resources that were used to produce the output. In the software development, the biggest part of whole production cost is the cost of work. Therefore, in the study, person hours were taken as the input unit of effort. It is much more difficult to select the metric for the unit of product. Two perspectives of measuring the size of the system can be identified:

- *Internal viewpoint* (developer's perspective)—describes the amount of code that must be produced to complete the system.
- *External viewpoint* (customer's perspective)—refers to the amount of functionality provided by the system.

The internal point of view metrics usually measure the physical 'length' of the code produced. Typical units of the internal size are number of source lines of code, number of classes or number of functions. Internal size measurements can be easily obtained by the use of automatic tools. The often mentioned weakness of measuring the system size using code lines is that result depends on the coding style—one programmer can write a statement in one line, while other can consistently spread it among a number of lines. To check if such a situation took place, the ratio of code lines per C++ statement was calculated for both projects. This metric should, to a certain extend, assure that coding style was similar in both projects.

The internal perspective may be confusing, since one platform may be 'more productive' when it provides a certain functionality using 'less code'. The measurement

from an internal perspective would not reveal this. However, the measurement from external perspective is difficult in real-time systems. This measurement should take into account not only the 'amount' of functionality but also the complexity, which is very difficult to quantify. Therefore, existing functional size metrics, like Function Points, are not recommended for real-time systems size measuring [26]. Instead of measuring, we decided to estimate the ratio of functional size of both systems. Since expert judgement is considered as an acceptable way of performing estimations [6,19,24], we used it for comparing the functionality of both systems.

The following data concerning the sizes of both projects were collected:

- *Number of person hours spent on each project (h)*—only the development phase of the project was taken into account (design, implementation, testing). Person hours contain designers, testers and managers work hours.
- *Number of code lines in each project (SLOC)*—we counted only lines with C++ code, comments and blank lines were not counted.
- *SLOC/C++ statement* ratios in both projects.
- *Number of classes in each project (NoC).*
- *Ratio of amount of the functionality in both projects (FUNC)*—an expert estimation. A total of six experts were interviewed for the estimation. All of them had knowledge concerning both projects. The results of the estimation were analysed by three other experts and a consensus was achieved.

3.2. Quality aspects

The main weakness of the size measurement methods is that they do not take any quality factors into account. The productivity can only be 'interpreted in context of overall quality of the product' [1]. Software product must meet certain quality requirements (minimum acceptable requirements) before the productivity metric can be applied. Therefore, in the study, quality aspects were kept in mind when evaluating productivity. Big differences in any aspect of quality may possibly explain the difference in productivity—in that case lower productivity could be the price for higher quality.

In the study, we have considered following quality factors that according to us can have impact on productivity:

- *Design quality*. Quality of application design and quality of code produced. Better design pays off in testing and maintenance phases and is more likely to be reused in other projects in future and therefore can be considered an added value.
- *Final product quality*. Qualities actually achieved in the final application. Example of such qualities may be non-functional requirements. High non-functional requirements (security, high availability) can be the important cost driver and therefore can explain relatively high development time.
- *Quality of development process*. High quality of development process does not guarantee high quality of final product, but makes achieving it more probable.

It is obvious that the lines of code are affected by a number of things. It is more difficult to produce well designed, structured and organized code. From the software metrics suggested by [9,11] that are applicable for object oriented systems, we selected those that were proven to have impact on quality of the system [2,7,8,11]. Therefore, a number of metrics, describing different aspects of design quality, were applied:

- *McCabe Cyclomatic Complexity (MCC)* metric [11]. This metric measures number of linearly independent paths through the function. According to [13] 'Overly complex modules are more prone to error, are harder to understand, are harder to test, and are harder to modify.'
- *Lack of Cohesion (LC)*. It measures 'how closely the local methods are related to the local instance variables in the class' [11]. The idea is to measure to what extent the class is a single abstraction. Chidamber et al. [8] proved that the Lack of Cohesion metric has impact on productivity. According to them the implementation of classes with high LC was difficult and time consuming. In the study, we counted LC using method suggested by Graham [14,17], which gives normalized values of LC (0–100%). We considered normalized values more applicable for comparison purposes.
- *Coupling (Coup)* [9,11], the metric measuring the number of classes the class is coupled to. This metric allows assessing the independence of the class. Coupling is widely recognized as important factor influencing productivity [8] and fault proneness [2,7].
- *Depth of Inheritance Tree (DIT)* measures how deep in inheritance hierarchy the class is. According to [9], high DIT values resulting in higher complexity make prediction of class behaviour more difficult. Basili et al. [2] proved that there is relation between DIT and fault proneness of the class.
- *Number of Children (NC)* defined as 'number of immediate subclasses subordinated to a class in the class hierarchy' [9]. Chidamber and Kemerer [9] claim that classes with huge amount of subclasses have potentially bigger impact on the whole design and therefore they require more testing (their errors are propagated).

Other quality aspects, like quality of the final product or quality of the development process, are difficult to quantify. Therefore, the opinions about both of them were collected during interviews. Twenty developers were interviewed in a semi-formal manner [22]; later an additional 10 informal

interviews were performed. When it comes to the quality of the final product, the interviews mainly concerned comparison of non-functional requirements put on the systems. The development process quality discussions focused on the amount of quality assurance activities like testing, inspections or the level of detail in project documentation.

3.3. Productivity bottlenecks

The next step after estimating the size of the productivity was the identification of issues that affect productivity. We were aiming at localization of productivity problems on the new platform. Therefore, we focused on identifying the productivity bottlenecks only in Project B. Since productivity can be affected by factors of different nature, the decision was made to take into account all the areas of productivity bottlenecks localizations, which are [15]:

- *People*. Issues connected with competence level of people involved in the development process.
- *Processes*. Issues connected with work organization characteristics.
- *Technology*. Issues connected with technology used; in our case mainly different platform shortcomings.

In order to identify the issues that affect the productivity, we performed interviews with 20 developers directly involved in the development on the new platform. The interviews were semi-formal [22]. Apart from reporting productivity bottlenecks, the interviewees were asked for suggestions of improvements/remedies. Based on data collected during the interviews, we created a list of productivity bottlenecks.

The additional analysis was performed to estimate to what extend each of the identified issues affects the productivity. The method selected for that is called Analytic Hierarchical Process—AHP [23]. Each respondent did pairwise comparisons between different issues and based on those comparisons the final importance of the different issues was calculated [23]. The comparison was based on the question 'Which alternative do you feel affects productivity more?'. The AHP questionnaires were distributed among the same group of people that took part in the interviews. The AHP method made it possible to build the hierarchy of the importance for each respondent and assigning weights of importance to alternatives. An example of importance vector for six issues is presented in Table 1. Such a vector is an outcome of an AHP analysis performed by a single respondent.

To ensure that the results really reflect the opinions of the respondents, each of them was presented with an individual importance vector and was allowed to change/adjust it. The individual importance vectors were used to create the importance vector of the whole group. It was created by calculating an average importance weight for each issue. The AHP analysis was performed by the same group of developers that took part in the interviews during which the productivity bottlenecks were identified.

Table 1
Example of the importance vector

Bottleneck	Importance (%)
Issue 1 (e.g. voice conversation)	30
Issue 2 (e.g. games)	10
Issue 3 (e.g. SMS)	20
Issue 4 (e.g. alarm)	15
Issue 5 (e.g. calculator)	15
Issue 6 (e.g. tunes)	10

N.B. importance figures always sum up to 100%.

4. Results

Each subsection in this section presents results of application of the methods described in corresponding subsection in Section 3.

4.1. Productivity measurement

Due to the agreement with the industrial partner, all measurement results will be presented either as [Project A/Project B] ratios or [Project A − Project B] differences. No results will be presented as absolute values.

The results of the size and the effort measurements taken on both projects are summarized in Table 2. The table presents relative values only.

To check if similar coding style was used in both projects, the average number of code lines/C++ statement was calculated for both projects. It turned out to be similar. In Project A it was 2,22 lines/statement, while in Project B it was 2,42 lines/statement. Therefore, we considered code lines comparable between both projects.

The significant difference (factor of 2) between the internal viewpoint measurement (code lines, classes) and the external viewpoint measurement (functionalities) suggests that UNIX is more successful in providing functionality—on average half of the code is required to provide a certain functionality. To explain that phenomena, the structure of both projects was examined. The structure is presented in Fig. 4. There is significant difference in the distribution of code in the subsystems. The large difference in the amount of code in the Platform part can easily be explained. In Project A, the own platform extension was developed on the top of the system, which was not the case

Table 2
Project size and effort ratios

Metric	Project A/Project B
Code lines (SLOC)	1.5
Number of classes (NoC)	1.5
Functional size (FUNC)	3.0
Person hours (h)	0.7

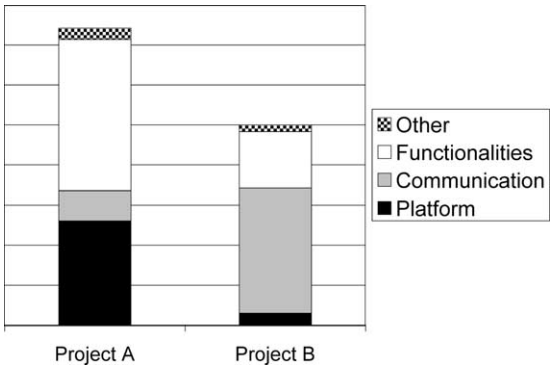

Fig. 4. Product structure.

in Project B. The subsystem responsible for the business logic (Functionalities) is about 3 times bigger in Project A which meets the expectations—according to the experts there is 3 times more functionality in Project A. In the unknown part there is a large difference in the amount of code for communication handling. We performed a number of interviews to explain the fact that more code is needed per average functionality. During the interviews we presented the diagram from Fig. 4. Three possible explanations of the phenomena were mentioned:

- In UNIX the support for communication is better—i.e. there are more third party libraries available or the system itself provides more.
- In general it is more difficult to design systems on the new platform. The design is more complex; more code has to be written to complete a certain functionality.
- The platform lacks certain tools (for example a good debugger) and therefore more code must be written to compensate for that (i.e. debug printouts that help in tracing faults).

From the information about the size and the effort, the development productivity ratios were calculated. The results are presented in Table 3.

4.2. Quality aspects

To compare the design quality in both systems, a number of measurements (described in Section 3.2) were done. Each metric will be analysed separately. Since all the metrics are done either on the function or on the class level (values are obtained either for each function or for each class) in order to compare two systems, we will examine the distribution of the values in each of them. For each metric we will present

Table 3
Productivity ratios

Metric	Project A/Project B
SLOC/PH	2.15
NoC/PH	2.15
FUNC/PH	4.30

Table 4
McCabe complexity

	Project A	Project B
Mean	3.3	2.5
Median	1	1
Maximum	125	96
Minimum	0	1
Std. deviation	5.8	5.0

mean, median, standard deviation and minimal and maximal values obtained. This way of describing measurements was presented in [2]. We will also present histograms describing in graphical form how many percent of the entities (classes, functions) in the system have certain value of the metric. Since it is sometimes difficult to assess if the obtained values are typical or not, where it is possible we will add a column where the corresponding values from the study described in [2] will be presented. Other examples of such values can be found in [7–9,20]. We selected values from [2] for comparison purposes because measurements there were taken on relatively large amounts of C++ classes, which is similar to our study, and the same types of data were collected as in our study. Values from other studies mentioned [7–9,20] will be used when discussing the findings.

The first metric applied was McCabe complexity. Results were obtained on function level and are presented in Table 4.

According to [11], McCabe complexity of the function should not be higher than 10, otherwise the function is difficult to test. We examined the data from that perspective (Fig. 5).

In both projects, the vast majority of the functions (95% in Project A and 97% in Project B) have complexity values within 0–10 range and therefore we consider both projects similar from that perspective.

The remaining measurements gave the results on the class level.

The second metric applied was the Lack of Cohesion. The results are summarized in Table 5.

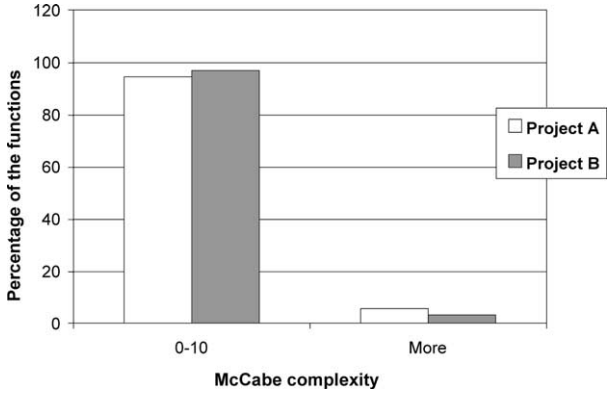

Fig. 5. McCabe complexity–distribution.

Table 5
Lack of cohesion

	Project A	Project B
Mean	49.9	46.8
Median	57	57
Maximum	100	100
Minimum	0	0
Std. deviation	35.5	37.4

Table 6
Coupling

	Project A	Project B	Ref. [2]
Mean	6.0	5.1	6.80
Median	3	3	5
Maximum	82	35	30
Minimum	0	0	0
Std. deviation	8.6	5.7	7.56

The distribution of LC values between classes of both systems is presented in Fig. 6.

We can observe that trends in distribution are similar in both systems. Mean and median values are also similar. Therefore, we consider both systems similar from a Lack of Cohesion viewpoint.

The next metric applied was Coupling. The results are summarized in Table 6.

In order to compare to what extend coupling values obtained in both projects are similar, we looked for data describing coupling in typical projects. The mean coupling values reported in [2,7–9,20] are usually between 5 and 7. The distribution of coupling values in the project classes is presented in Fig. 7.

We consider the distribution of coupling values similar for both projects. The values from Table 5 place both our projects among typical projects from coupling point of view. Therefore, we consider them similar from that perspective.

The application of the Depth of Inheritance Tree metric gave results presented in Table 7.

Compared to [2,9], the mean, median and standard deviation values obtained in the study are much lower. The distribution of DIT values is presented in Fig. 8.

From Fig. 8, it can be observed that the difference between projects is caused by about 20% of the classes that in Project B have depth 1, while in Project A the depth is 0. We consider that difference rather small especially because compared to other studies the average DIT values in both examined projects are small.

The last metric applied was Number of Children (NoC). The results are summarized in Table 8.

Compared to the reference project, we can observe that Project A has similar characteristics, while in Project B mean value is about twice as small. The median value is in both cases equal to 0. The distribution of NoC values is presented in Fig. 9.

As it can be seen in Fig. 9, over 90% of classes in both projects has NoC equal to 0. The difference in mean value is caused mostly by 2% of classes that in Project A have 1 child, while in Project B they have 0. Since the distribution is almost identical, we consider both projects similar from NoC perspective.

After analysing all the measurements taken, we cannot observe any major difference in design quality between the two projects. Therefore, we consider the design quality similar in both projects.

The measurements presented above describe only design quality. Equally important for the productivity assessment are the quality aspects of the system developed (i.e. non-functional requirements) and the quality aspects of the development process (i.e. number of the quality assurance activities). The overall impression was that Project B was more ambitious in terms of the process quality aspects. Due to the relative novelty of the platform, and in order to minimize anticipated influence of the learning effect, much effort was put on quality assurance (inspections, testing) and documentation activities. The non-functional requirements put on the systems were similar, but during interviews the designers mentioned that they believed the non-functional characteristics achieved in the project were better in the system developed on the new platform.

Both systems met the requirements that were put on them.

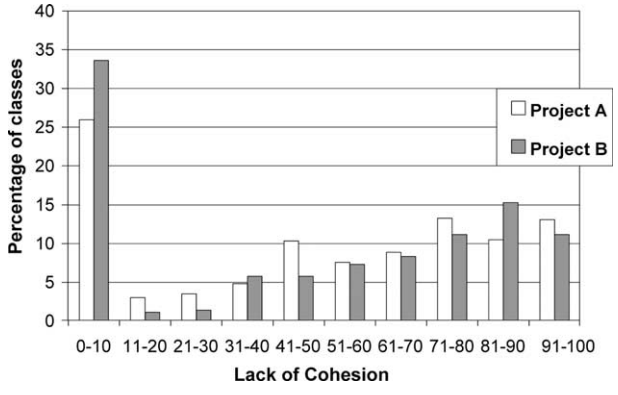

Fig. 6. Lack of cohesion—distribution.

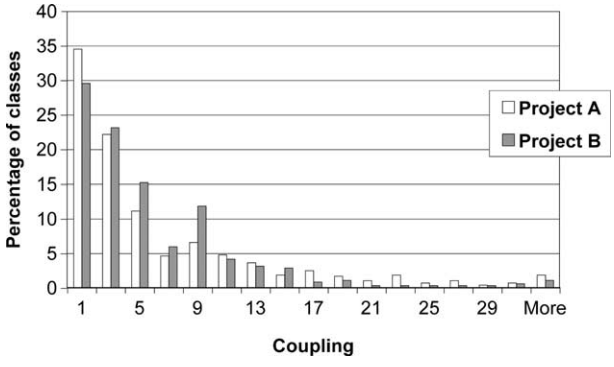

Fig. 7. Coupling—distribution.

Table 7
Depth of inheritance

	Project A	Project B	Ref. [2]
Mean	0.62	0.78	1.32
Median	0	1	0
Maximum	4	3	9
Minimum	0	0	0
Std. deviation	0.8	0.7	1.99

Table 8
Number of children

	Project A	Project B	Ref. [2]
Mean	0.35	0.19	0.23
Median	0	0	0
Maximum	24	9	13
Minimum	0	0	0
Std. deviation	1.8	0.9	1.54

4.3. Productivity bottlenecks

The nature of the productivity problems was identified during interviews. As a result of them, the list of nine issues that negatively affect the productivity in the Project B was formulated:

- *Not enough experience sharing* (*i.e. seminars*, *meetings*) *and training activities*. This problem impacts the productivity in two ways:
  - The skills of the staff are not developed as quickly as it would be possible.
  - The 'reuse' of ideas is smaller. Better exchange of the information would prevent the situation when two people invent a solution to the similar problem separately.
- *Staff competence level*. The novelty of the platform causes overhead connected with learning. This is important because, according to interviewees, the start up time on the new platform is relatively long compared to, e.g. UNIX.
- *Quality of the new platform's documentation*. Certain problems connected with the documentation's quality and availability were mentioned. According to interviewees, a better structured and updated source of technical information on the new platform would positively affect the speed and quality of software development.
- *Runtime quality of the new platform*. Low runtime quality of the platform makes development and testing more difficult. Relatively, high amount of faults was classified as platform related, which means that they occurred due to the platform error. Therefore, the scope of potential fault localizations is bigger compared to more mature platforms, which adds a lot of complexity to testing.
- *The platform interface* (*API*) *stability*. Unexpected changes in the API in the different platform releases result in the need of 'redoing' parts of the system to meet the new API specification.
- *Lack of target platform in the design phase*. Designers do not have access to a real cluster. Instead, they are working with an emulator, which is not 100% compatible with the target platform. The faults caused by the incompatibility are discovered later, in testing phase, when the cost of correction is much higher.
- *Too much control in the development process*. Due to the novelty of the platform there was a strong pressure on the quality assurance activities, like large amount of inspections or detailed documentation produced on quite low level. It resulted in heavy and costly development processes.
- *Unstable requirements*. The unstable requirements make it necessary to redesign parts of the system, sometimes in later stages of the project, which is extremely costly.
- *Too optimistic planning and too big scope of the projects*. Big scope of projects combined with the novelty of the platform may result in long lead-time. Long lead-time projects are much more prone to change requests due to changes of market demands.

It is noticeable that the bottlenecks identified have different nature. This suggests that the productivity problem is complex and depends on many factors.

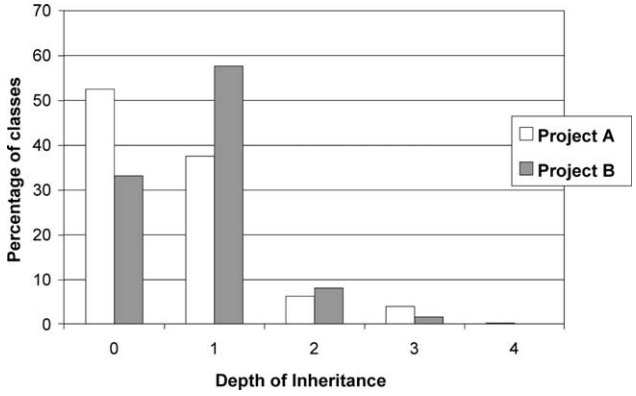

Fig. 8. Depth of inheritance—distribution.

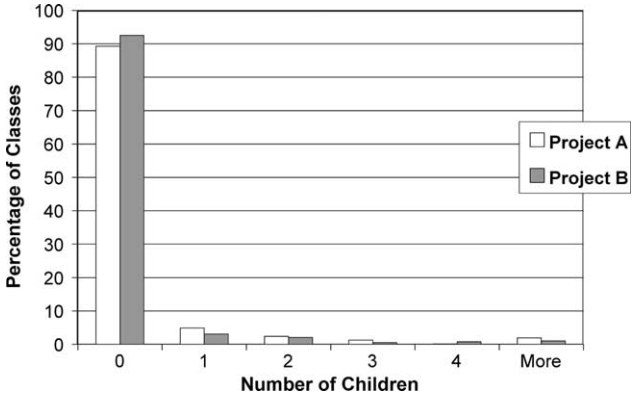

Fig. 9. Number of children—distribution.

Table 9
Productivity bottlenecks prioritization

Bottleneck	Importance (%)
Staff competence level	22
Unstable requirements	16
Not enough experience sharing and training activities	13
Quality of platform's documentation	10
Runtime quality of the platform	10
Too much control in development process	9
Too optimistic planning, too big scope of projects	8
Lack of target platform in the design phase	7
Platform interface stability (API)	5

The interviewees were asked to prioritize the issues to show which, according to them, affects productivity most. Thanks to the use of the AHP method we were able to create individual importance vectors. For each interviewee, we calculated the weights of importance that the interviewee assigned to the issues. Weights were normalized, so they always sum up to 100%. Based on the individual importance vectors, the average vector was calculated (see Section 3.3). Table 9 presents the final ranking of the issues affecting the productivity of the software development on the new platform.

5. Discussion

5.1. Related work

In the literature, the software productivity research has gone in two main directions [21]:

- productivity measurement;
- identification of factors that affect productivity.

Main point of concern of researchers dealing with measurements is the lack of a generally accepted metric for measuring software size. Often the simplest method of measuring the size is used, which is number of lines of code [3,4,21,30]. Maxwell et al. [21] performed a study that involved an evaluation of the lines-of-code productivity metric. They compared it with Process Productivity, the complex metric that includes management practices, level of programming language, skills and experience of the development team members and the complexity of the application type. After examining 99 projects, the authors concluded that the simple lines-of-code metric was superior to the Process Productivity metric.

The second direction of the productivity research was identification of factors that affect the productivity. The continuously increasing cost of software development has made productivity improvement a very popular topic. It is usually seen as the way to decrease cost and improve delivery time. The fact that software projects are often late and over budget [3,4,28] makes the problem important. A number of research studies were done within that domain. Yu et al. [30] created a universal framework for the improvement process. They identified three steps of the improvement process: measurement, analysis and improvement. The purpose of the first step was to find out where the project stands, the second step was devoted to identification of the factors affecting productivity and the third one was supposed to minimize their impact. The framework suggested corresponds well with our strategy. The authors presented the example of the study aiming into quantification and improvement of the productivity in a project from AT&T Bell Laboratories. One interesting finding in that project is that the issues affecting productivity are, to a large extent, similar to the ones described in our research. Among the issues that were ranked, the highest belong to the requirements stability and the staff experience, which is similar to the results obtained in our research.

Many researchers have observed, documented and tried to solve the productivity problem when adopting new technology [10,12,16,18,29]. Ever since the learning effect [25] was described most researchers agree that some of the initial productivity problems fade away with time due to growing experience and maturity of the organization. The question of how to make that time as short as possible remains unanswered. Edmondson et al. [10] stress the importance of having as much of the knowledge about new technology codified as possible. The more of the knowledge about new technology is tacit; the bigger problem is to introduce it seamlessly. In the examined project, we have experienced the situation where lack of easily accessible source of information about the technology affected the productivity. Fisher and Wesolkowski [12] present another view of the initial knowledge problem. It might be extremely difficult to increase the competence level of the staff if the staff does not want it. According to them, one of the key points is the motivation and attitude issue—they quote the results of the study showing that only 15% of the population is enthusiastic when it comes to new technology adoption, 85% is more or less hesitant to it. Harvey et al. [16] analysed 100 companies to find out how the companies that successfully adopted new technologies differed from the others. One of their findings was the huge role of management in such process. The importance of management's role in facilitating the process of new technology adoption is also stressed by Vaneman and Trianfis [29]. Among the bottlenecks we have identified there are issues concerning project planning and organization, which proves that the role of management was recognized by our interviewees.

Other researchers also put a lot of effort in localizing issues affecting the productivity. Productivity issues are often referred to when discussing delays in software deliveries [3,4,28]. Although lead-time and productivity do not always correlate (i.e. adding a new developer may decrease the productivity, but improve lead-time), the improvement of productivity is usually seen as a way to

decrease the development time. Blackburn and Scudder [3] identified number of factors that reduce development time. The authors examined the data from 40 different projects. As the most promising technique of development-time reduction, the Reuse of Code was considered. The second most promising technique was competence development, which is similar to the results obtained in our study. Moreover, in the paper, the authors report that 'managers are continually frustrated by changing requirements'—which directly corresponds to 'Unstable requirements', the issue which is second on our priority list.

A direct comparison of results obtained in the research studies presented above with results obtained in our study may seem to be inappropriate—these were usually surveys in which a large number of projects were examined, and therefore the results are on higher level of abstraction and to large extend their generalization would be justified. However, the fact that conclusions concerning productivity bottlenecks seem to be similar may indicate that the problems identified are rather common for the situation when new technology is adopted.

5.2. Productivity level

The results obtained in the study describe the current productivity level in the software development on the new platform. Compared to the UNIX platform, a factor of four in the productivity measured from an external perspective was identified. It can be later decomposed into two issues:

- *Code writing speed*. On the new platform, the code is written slower, on average it takes twice as much time to deliver 1 line of code (internal perspective measurement measures the speed of delivering the code).
- *Code line/functionality*. In the new platform, the average number of code lines per functionality is bigger. Since it holds the remaining part of responsibility for the productivity level, its impact may be counted as a factor of two. It is supported by measurements—on average we need twice as much code per functionality on the new platform.

The issues that affect code writing speed are presented in Table 9. The ones that have impact on high SLOC/functionality ratio in the new platform are lack of third party libraries, missing tools (e.g. debuggers) and platform complexity. The distribution of responsibility for productivity level is summarized in Fig. 10.

5.3. Productivity improvement

The main reason for bottlenecks identification is to find out where the application of remedies would bring the best results. In our study it seems obvious, since three highest ranked issues hold over 50% of responsibility for the productivity level, and two of them are related to

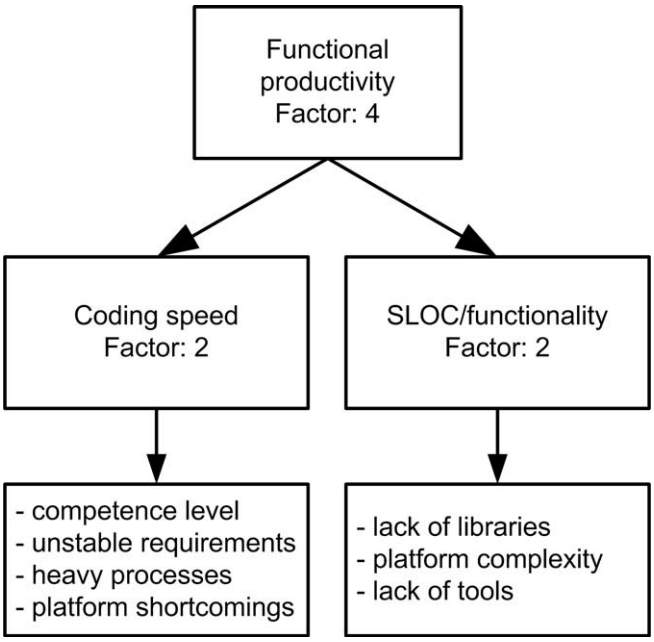

Fig. 10. Productivity problem decomposition.

competence. It is not surprising—competence is usually a problem when new technology is introduced, especially, a complex one with long start up time. The picture changes if localizations of bottlenecks, suggested by Hantos and Gisbert [15], are considered. Table 10 presents this.

A surprising finding is that platform related issues were rated quite low both individually and as a group. One possible explanation could be that the respondents focused on issues that are internal to the organization, where the study was performed. Maybe, they tried to focus on issues that directly depend on them. Platform quality issues do not belong to that group of issues, while competence and work organization issues do. Another explanation would be more straightforward—platform quality is not the main problem. Additionally, it should be noticed that the competence issues' importance is most prone to change over time. It should decrease with time when the developers will gain

Table 10
Bottleneck localizations importance

Localization	Bottlenecks	Importance (%)
People	Staff competence level Not enough experience sharing and training activities	35
Processes	Unstable requirements Too much control in development process Too optimistic planning, too big scope of projects Lack of target platform in design phase	40
Technology	Quality of platform's documentation Runtime quality of the platform Platform interface stability (API)	25

experience. However, it is still a valid issue when the platform is being introduced to a new organization.

Table 10 clearly suggests that each group of the issues holds relatively large responsibility for current productivity level—no 'silver bullet' solution to the problem can be expected. Therefore, remedies to all the productivity bottlenecks were presented. It is a subject to further research to suggest the order in which they should be applied. Their possible effectiveness, estimated in this study, is only one of the factors that should be taken into consideration when making that decision—others are the cost of the remedy, risk connected with its introduction or the time after which the remedy application will pay off.

There are three ways to improve productivity [5]: work faster, work smarter and work avoidance. Faster work can be obtained by development of skills. Therefore, the following skill development activities were suggested:

- *Good introduction process*. The introduction process would familiarize the staff with the new technology and minimize the overhead connected with learning.
- *Continuous skills development processes*, like an advanced course on programming on the introduced platform, seminars, meetings and technical discussions would give the developers a chance to share experiences and spread knowledge among team members.
- *Better management of company knowledge*
  - Set of patterns—set of easily applicable solutions to the common problems.
  - Better documentation of the problems encountered would help to avoid making the same mistakes in the future.

The second way of improving productivity, smarter work, suggests better work organization. The following possible remedies were suggested to the problems identified:

- *Lack of target platform in the design phase*
  - More automated functional tests run overnight would provide immediate feedback concerning the application's behaviour on the real platform. The faults would be detected earlier, which would save a lot of time connected with, e.g. fault localization.
  - Introduction of the target platform in the analysis/design phase would be an expensive solution, but would provide designers with immediate and precise feedback.
  - More prototyping in the early stages of the project—some problems that are encountered in the implementation phase would never occur if the ideas were tested on prototypes earlier in the design phase.
  - Shorter time between functionality development and testing would result in faster fault detection. It would be easier for designer to localize the problem in code that was recently produced than in code produced long time ago.
- *Too optimistic planning, too big scope of projects and unstable requirements*—a smaller scope of the projects would result in more stable requirements. The main cause of requirements change is the change of market demands. If the project had shorter lead-time, the probability of having change requests would be smaller and their impact would be minimized.

The last way of productivity improvement, namely 'work avoidance', refers to the idea of acquiring the solution instead of developing it. One big issue in that topic is the current lack of third party components for typical purposes, like handling of the standard communication protocols. Other issues are the platform quality issues. The following improvements of the platform were suggested to the platform developer:

- *Runtime quality of the platform*. The focus should be put on providing quality to existing functionalities of the platform instead of developing new features.
- *Platform programming interface* (*API*) *stability*. A 'roadmap' describing which parts of the platform are subject to change would solve the problem. In that case, the designers would not be surprised by API changes.
- *Quality of platform's documentation*—update documentation. Features that are not documented cannot be used, so there is no point in developing new features if their description is not added to the documentation.

6. Conclusions

The objective of the study was to examine the impact of the change of the platform on the software development from the development productivity point of view. To achieve that we at first quantified the productivity of the software development on the introduced platform, then identified the nature of the problems encountered and finally we suggested some productivity improvement methods. The quantification was done by comparison with an other project, in which the productivity was perceived as good.

The measurement from the functional perspective revealed the factor of four difference between the development productivity in the two projects. Examination of the product structure and measurement from the internal perspective allowed us to select two factors that result in the factor of four differences between productivity. These are the code writing speed, twice as small in the new platform, and the average amount of code necessary to provide certain functionality, about twice as big in the new platform.

In order to check to what extend the difference in productivity between the two projects could have been

caused by the difference in quality, we examined different aspects of quality. We found out that in terms of the design quality both projects are similar, but in terms of process quality the project done on the new platform was more ambitious. In terms of non-functional requirements there was no significant difference in the requirements put on the system, but the developers believed that the non-functional characteristics actually achieved in the project done on the new platform were better.

Later, we identified factors that affect productivity and we estimated their importance. It turned out that the problem is complex—there are many factors of different origins that affect the current productivity level. The learning effect, caused by the introduction of the new platform, had the relatively highest impact on the code writing speed. However, other factors like the platform quality and the work organization also have a significant impact. Lack of third party libraries for the new platform and the platform complexity result in larger amount of code necessary to deliver functionality, compared to the UNIX environment.

Due to the problem complexity, the suggestion of one, 'silver bullet' solution was impossible. Therefore, we suggested a number of remedies to the issues identified. The individual importance of the issue the remedies address will be one of the factors taken into consideration when deciding the order in which the remedies will be applied.

We believe that some general lessons can be learned from our study. Problems, similar to the ones we have described, may be experienced every time a company decides to change platform or technology. If the change is from a standard, widely used environment, to one used mainly in specialized application domains, as in our case, it seems very likely that the bottlenecks we have identified may appear. The magnitude of their impact may, however, differ, due to their dependence on individual settings like the kind of technology introduced, experience of the staff, characteristics of projects done in the company and many others.

One of the main issues identified in our study, namely the learning effect, is always present when a new platform is introduced. If the platform has a unique programming model, the learning curve can be very steep. Appropriate training activities, although expensive, may bring significant savings on the project cost. This was clearly pointed by our interviewees who ranked the competence issues highest. Therefore, it seems to be extremely important that the developers are provided with a good source of information about the new platform. It not only minimizes the learning effort, but also affects the coding speed even after the developers have gained a certain level of experience.

Due to the learning effect projects done using the new technology are prone to delays. In order to minimize the impact of limited experience on the quality of the product, often the amount of quality assurance activities is higher than normally, which makes the project even more delayed. In the systems like the ones we have examined, it immediately results in unstable requirements, which according to our interviewees have significant impact on productivity. Therefore, the scope of initial projects should be limited, if possible.

Another issue that may be a consequence of introduction of a very specialized platform is the lack of convenient add-ons that are available on standard, widely used platforms. The number of available tools or third party libraries is likely to be limited since the relatively small number of potential customers that would buy those makes their development questionable from economical perspective. Considering that among those, there are debuggers, profilers or CASE tools as well as software libraries; the impact of their absence should not be underestimated.

Acknowledgements

The authors would like to thank all the members of Ericsson's staff thanks to whose participation, patience and will to help the whole study was possible as well as Daniel Häggander and Johan Schubert from BTH for their help, contribution and valuable comments.

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project 'Blekinge-Engineering Software Qualities (BESQ)' (http://www.ipd.bth.se/besq).

References

[1] IEEE Standard for Software Productivity Metrics, IEEE Std, 1045–1992, 1993.
[2] V.R. Basili, L.C. Briand, A validation of object-oriented design metrics as quality indicators, IEEE Transactions on Software Engineering 22 (1996) 751–762.
[3] J.D. Blackburn, G.D. Scudder, Time-based software development, Integrated Manufacturing Systems 7 (1996) 60–66.
[4] J.D. Blackburn, G.D. Scudder, L.N. van Wassenhove, Improving speed and productivity of software development: a global survey of software developers, IEEE Transactions on Software Engineering 22 (1996) 875–886.
[5] B. Boehm, Managing software productivity and reuse, Computer 32 (1999) 111–114.
[6] B.W. Boehm, Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[7] L.C. Briand, et al., Investigating quality factors in object-oriented designs: an industrial case study, Proceedings of the International Conference on Software Engineering 1999; 345–354.
[8] S.R. Chidamber, D.P. Darcy, C.F. Kemerer, Managerial use of metrics for object-oriented software: an exploratory analysis, IEEE Transactions on Software Engineering 24 (1998) 629–639.
[9] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (1994) 476–494.
[10] A.C. Edmondson, et al., Learning how and learning what: effects of tacit and codified knowledge on performance improvement following technology adoption, Decision Sciences 34 (2003) 197–223.

[11] N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS, London, 1997.
[12] W. Fisher, S. Wesolkowski, How to determine who is impacted by the introduction of new technology into an organization. Proceedings of the 1998 International Symposium on Technology and Society, ISTAS 98, Wiring the World: The Impact of Information Technology on Society, 1998, pp. 116–122.
[13] S. Gascoyne, Productivity improvements in software testing with test automation, Electronic Engineering 72 (2000) 65–67.
[14] I. Graham, Migrating to Object Technology, Addison-Wesley, Reading, MA, 1995.
[15] P. Hantos, M. Gisbert, Identifying software productivity improvement approaches and risks: construction industry case study, IEEE Software 17 (2000) 48–56.
[16] J. Harvey, L.A. Lefebvre, E. Lefebvre, Exploring the relationship between productivity problems and technology adoption in small manufacturing firms, IEEE Transactions on Engineering Management 39 (1992) 352–359.
[17] B. Henderson-Sellers, L.L. Constantine, I.M. Graham, Coupling and cohesion (towards a valid metrics suite for object-oriented analysis and design), Object Oriented Systems 3 (1996) 143–158.
[18] M. Huggett, S. Ospina, Does productivity growth fall after the adoption of new technology?, Journal of Monetary Economics 48 (2001) 173–195.
[19] R.T. Hughes, Expert judgement as an estimating method, Information and Software Technology 38 (1996) 67–76.
[20] X. Li, et al., A measurement tool for object oriented software and measurement experiments with it, Tenth International Workshop New Approaches in Software Measurement, Springer, Berlin, 2001, pp. 44–54.
[21] K.D. Maxwell, L. Van Wassenhove, S. Dutta, Software development productivity of European space, military, and industrial applications, IEEE Transactions on Software Engineering 22 (1996) 706–718.
[22] C. Robson, Real World Research: A Resource for Social Scientists and Practitioner-Researchers, Blackwell, Oxford, UK, 2002.
[23] T.L. Saaty, L.G. Vargas, Models, Methods, Concepts and Applications of the Analytic Hierarchy Process, Kluwer, Boston, 2001.
[24] M. Shepperd, M. Cartwright, Predicting with sparse data, IEEE Transactions on Software Engineering 27 (2001) 987–998.
[25] G.J. Steven, The learning curve: from aircraft to spacecraft?, Management Accounting 77 (1999) 64–65.
[26] C.R. Symons, Software Sizing and Estimating: Mk II FPA (Function Point Analysis), Wiley, Chichester, 1991.
[27] P. Tomaszewski, L. Lundberg, Evaluating Productivity in Software Development for Telecommunication Applications, The IASTED International Conference on Software Engineering, IASTED, Innsbruck, Austria, 2004.
[28] M. van Genuchten, Why is software late? An empirical study of reasons for delay in software development, IEEE Transactions on Software Engineering 17 (1991) 582–591.
[29] W.K. Vaneman, K. Trianfis, Planning for technology implementation: an SD(DEA) approach, PICMET '01, Portland International Conference on Management of Engineering and Technology, Technology Management in the Knowledge Era., PICMET—Portland State University, Portland, OR, USA, 2001.
[30] W.D. Yu, D.P. Smith, S.T. Huang, Software productivity measurements, Computer Software and Applications Conference, COMPSAC '91, Proceedings of the 15th Annual International, 1991 pp, 558–564.