Information and Software Technology 56 (2014) 477–494

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Systematic analyses and comparison of development performance and product quality of Incremental Process and Agile Process

Ayca Tarhan [*], Seda Gunes Yilmaz [1]

Hacettepe University Computer Engineering Department, Beytepe Kampusu, 06532 Ankara, Turkey

ARTICLE INFO

Article history:
Available online 16 December 2013

Keywords:
Empirical method
Quantitative analysis
Qualitative analysis
Software measurement
Process performance
Agile development

ABSTRACT

Context: Although Agile software development models have been widely used as a base for the software project life-cycle since 1990s, the number of studies that follow a sound empirical method and quantitatively reveal the effect of using these models over Traditional models is scarce.
Objective: This article explains the empirical method of and the results from systematic analyses and comparison of development performance and product quality of Incremental Process and Agile Process adapted in two projects of a middle-size, telecommunication software development company. The Incremental Process is an adaption of the Waterfall Model whereas the newly introduced Agile Process is a combination of the Unified Software Development Process, Extreme Programming, and Scrum.
Method: The method followed to perform the analyses and comparison is benefited from the combined use of qualitative and quantitative methods. It utilizes; GQM Approach to set measurement objectives, CMMI as the reference model to map the activities of the software development processes, and a predefined assessment approach to verify consistency of process executions and evaluate measure characteristics prior to quantitative analysis.
Results: The results of the comparison showed that the Agile Process had performed better than the Incremental Process in terms of productivity (79%), defect density (57%), defect resolution effort ratio (26%), Test Execution V&V Effectiveness (21%), and effort prediction capability (4%). These results indicate that development performance and product quality achieved by following the Agile Process was superior to those achieved by following the Incremental Process in the projects compared.
Conclusion: The acts of measurement, analysis, and comparison enabled comprehensive review of the two development processes, and resulted in understanding their strengths and weaknesses. The comparison results constituted objective evidence for organization-wide deployment of the Agile Process in the company.

1. Introduction

Software development activities such as requirements analysis, design, implementation, testing, and maintenance are carried out within a software development life-cycle in an organization [1]. The life-cycles are assumed to be adapted from software development models that have been implemented for years and proved successful. However, many software projects adapting different development models for their life-cycles fail to achieve their targets. In accordance to an analysis performed by The Standish Group in 2009 [2], only 32% of software projects were reported as successful in comparison to reported 44% as challenged (late, over budget and/or with less than the required features and functions) and 24% as failed (cancelled prior to completion or delivered and never used).

Low success rates of software projects motivated inquiry of Traditional (Plan-driven) software development models in 1990s and in the following years, Agile software development models were proposed as an alternative. The Traditional models were defined to value fully specified problems, rigorous planning, pre-defined processes, and regular documentation [1]. The Agile models, on the other hand, were defined to value individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan [3]. Although Agile software development models have been widely used as a base for the software project life-cycle since 1990s, the number of studies that quantitatively reveal the effect of using these models over Traditional models on development performance is scarce. In their systematic review on empirical studies of Agile software development, Dybå and Dingsøyr [4] claim the immaturity of the methods,

* Corresponding author. Tel.: +90 312 2977500; fax: +90 312 2977502.
E-mail addresses: atarhan@cs.hacettepe.edu.tr (A. Tarhan), sedagunes@gmail.com (S.G. Yilmaz).
1 Tel.: +90 312 2977500; fax: +90 312 2977502.

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.12.002

the data, and the results in the limited number of reported studies. Petersen and Wohlin [5] emphasize the need to investigate Agile and incremental models using sound empirical methods, and report a case study revealing the effect of moving from a Plan-driven to an Incremental and Agile development approach. Van Waardenburg and van Vliet [6], on the other hand, investigate the challenges that are brought by the co-existence of Agile models and Plan-driven development and rationalize how two organizations deal with those challenges.

Quantitatively revealing the effect of using software development models on development performance and product quality in various contexts is among the major needs of the software community. However, a systematic review on measurement in software engineering by Gómez et al. [7] claims that software process is the least measured entity (with 21%) in the reported studies. This is mostly due to the diverse issues to consider while empirically measuring and analyzing the performance of a software development process. In addition to having knowledge on process management, measurement, and statistics, this work requires a series of tasks to be carried out such as; identifying the purpose of the analysis, capturing process context, identifying process components, ensuring consistency of process executions, gathering process data, selecting process measures, determining analysis methods, conducting the analysis, and interpreting analysis results. When comparing the performances of two or more development processes, reconciling the scopes of the processes as a base for the comparison is another task to accomplish. Aside from these challenges, there are only a few systematic and practical methods [8,9] that can be utilized as a guide while measuring the performance of a software development process.

In this paper, we explain the steps of and the results from systematic analyses and comparison of development performance and product quality of Incremental Process and Agile Process that are both adapted in a middle-size, telecommunication software development company employing 65 developers. The Incremental Process which is in use for 16 years is an adaption of the Waterfall Model whereas the newly introduced Agile Process is a combination of the Unified Software Development Process, Extreme Programming, and Scrum. The projects in which the Incremental and Agile processes were employed have been carried out to create a product family, and the development of 20 features in each project were taken as the base.

The method followed to perform the analyses and comparison proposes the utilization of qualitative and quantitative methods together to derive the conclusions. This paper describes the empirical method in detail and presents example outputs from its application, which is hardly met in the literature. The method utilizes; Goal Question Metric (GQM) Approach [8] to set performance and quality measurement objectives, Capability Maturity Model Integration (CMMI) for Development [10] as the reference model to map the activities of the software development processes, and a pre-defined assessment approach [11] to verify consistency of process executions and evaluate measure characteristics prior to quantitative analysis. The GQM Approach is taken as a guideline to identify measurement goals and required measures, and complemented by a bottom up approach in order to reconcile required measures with available process measures.

The article is organized in seven sections. Section 2 provides background information on the Traditional and Agile software development models, software measurement basics and the GQM Approach, the Capability Maturity Model Integration, the pre-defined assessment approach called an Assessment Approach for Quantitative Process Management, and the related work on comparative studies of software development models. Section 3 provides an outline of the overall case study including context and scope, research objective and design, and analyses goals and measures. Section 4 describes the analysis method and demonstrates example outputs from the analysis of the Incremental Process. Section 5 presents the quantitative results of the comparison, and discusses the rationale behind these results based on the qualitative data elicited from process enactments of the Incremental and Agile processes. Section 6 elaborates threats to validity for the case study and Section 7 provides the conclusions.

2. Background

2.1. Software development models

The organization of the processes and activities that are used to develop a software product is known as "software development life cycle (SDLC)". The SDLC is aimed to ensure systematic and purpose-oriented development of software products. There are several models that describe how to build this cycle under Traditional and Agile approaches.

In the Traditional approach, process activities are planned in advance and progress is measured against this plan [1]. In the Agile approach, on the other hand, planning is incremental and it is easier to change the process to reflect changing customer requirements. Agile models are aimed to respond to changing requirements, to swiftly produce software, and to make these productions available for the customers [12]. Table 1 provides a summary of the characteristics of the Traditional and Agile models.

2.2. Software measurement and the Goal Question Metric (GQM) Approach

Measurement is vital to understanding, controlling, and improving in an engineering discipline. In software engineering, however, measurement is considered a luxury many times [23]. Because of the abstract and changing nature of the software and the pressure in the timely completion of the software product, management in software engineering is mostly based on observation and assumption, rather than on objective evidence. Creating objective evidence, on the other hand, is not easy and is enabled by adequate resources and training, and by appropriate use of methods and tools. This subsection, therefore, provides an overview of measurement, software measures, and the recommended methods.

Measurement is the process by which numbers and symbols are assigned to attributes of entities in the real world, as to describe them according to clearly defined rules [23]. A measure must specify the domain and the range as well as the rule for performing the measurement mapping. Both entity and attribute to measure should be explicit. Measures can be direct or indirect [23]. Direct measures involve no other attribute or entity, and form the building blocks for assessment. Examples are size, duration, and number of defects. Indirect measures are derived from other measures. Examples include productivity, defect density, and efficiency.

Measurement mapping together with the empirical and numerical relation systems represent the measurement scale [24]. Scales help us to understand which analyses are appropriate on measurement data. Types of scales are nominal, ordinal, interval, ratio, and absolute, in the increasing order of information provided [24]. Nominal scale indicates a difference, just classification, and no ordering (e.g., names of programming languages). Ordinal scale indicates the direction of the difference, and ranking with respect to ordering criteria (e.g., priority assignments of defects). Interval scale indicates the amount of the difference; thus, differences of values are meaningful (e.g., calendar date). Ratio scale indicates an absolute zero; therefore, ratios between values are meaningful (e.g., effort). Absolute scale indicates the number of values (e.g.,

Table 1
Characteristics of Traditional and Agile software development models.

Development model	Basic characteristics	Refs.
Traditional (Plan-driven)		
Waterfall	– simple, linear, ordered activities – regular and heavy documentation – activities include feasibility study, requirements analysis, design, implementation and unit test, integration and system test, and operation and maintenance – once an activity is completed, it is difficult to return back to it – beneficial when requirements can be clearly and completely identified at the start of the project and the changes to these requirements can be restricted thereafter	[1,13]
Incremental	– a modified version of the Waterfall Model – software is developed incrementally in parts as a response to increasing product sizes – the initial increment is a core product part with urgent, high-priority functionality; and following increments are delivered to the customer in sequence, each with a wider functionality – beneficial when requirements are incomplete at the start of the project and can be elicited in time	[13]
Evolutionary	– supports the identification of functionality to be developed with the customer – the first functionality corresponds to a major part of the product and is developed and delivered to the customer in one round – the success of the project is evaluated based on this first evolution – if succeeded, additional requirements are handled in the following evolutions – beneficial when projects include technological uncertainties at the start and the majority of the product functionality is to be delivered in a lump	[1]
Spiral	– based on risk analysis and prototype development – the phases –cycles– of the spiral are identified in accordance to project requirements – the spiral is divided into four quarters: Planning, risk analysis, production, and user evaluation – at each phase, the risks are identified, and a prototype planned for that phase is implemented; and at the end of the phase, objectives are evaluated, and alternatives and constraints are identified – its relatively complex structure requires expertise for adaptation – beneficial when projects are large and critical	[14]
Agile		
Extreme Programming	– comprises of practices which enable the strength of customer relations and team interactions, and more feedback – traits include small releases, swift feedback, customer participation, interaction and concordant working, continuous integration and test, mutual code ownership, limited documentation and pair programming – beneficial when requirements are instable and constantly changing	[15]
Feature-Driven Development	– works in accordance with the features generated from software requirements – prior to bringing a new feature into the system, the structure covering this feature is formed – steps include; the development of overall system model, listing the features, planning in a feature-oriented way, the production of feature-oriented design, and making feature-oriented development – check out rules of every step is defined and obeyed – beneficial for critical systems development	[16]
Crystal Family of Methodologies	– regards adaptability to the project, interaction-oriented mentality, and human force – has two principles: (1) make increments which last one-to-three months, (2) organize seminars where successes and failures are discussed prior to and following each increment – its methods are labeled as white, yellow, orange, and red according to the difficulty; the more the color is darker, the more the method is harder – the method with proper color is selected according to the project's extent and criticality	[17]
SCRUM	– focuses on practices about project management and does not include engineering details – postulates that the project scope is to change during the project – has practices which can evade instability and complications instead of having specific practices of software development (e.g., every executable increment of the software called "sprint" is completed within 30 days and delivered to the customer)	[18]
Dynamic System Development	– enables the control of the steps of the short term and fast software development – the convenience of the use of the model is determined by feasibility study and business study – the rest of the process is carried out by such steps as functional model iteration, system design and build iteration, and implementation – principles include; the changeability of project requirements according to project sources and calendar, frequent deliveries, and implementation of tests at every stage of the project	[19]
Lean Software Development	– based on the lean manufacturing principles used in the reorganization of Japanese automobile industry – has seven principles: Eliminating waste, amplifying learning, deciding as late as possible, delivering as fast as possible, empowering the team, building integrity in, and seeing the whole – enables the change to create new opportunities by controlling the change with restrictive management implementations	[20,21]
Unified Development Process	– an architecture-centric, use-case driven method – there is continued debate about whether it can be classified as Agile – has artefacts that are designed to eliminate variance and improve estimation ability, and it can be possible to reduce the lead time and expenses by the iterations	[19,22]

number of defects). The following statistical operations are allowed – including the operations of the previous scales – for each scale [24]: Mode and frequency in Nominal; median and percentile in Ordinal; mean and standard deviation in Interval; geometric mean in Ratio; counting in Absolute. A mapping from one acceptable measure to another is called an admissible transformation [23]. The transformations do not change the structure of the scale (e.g., feet mapped to inches).

There are several measures widely used in software engineering to measure process, products, and resources. Basics of these include defect density, verification and validation (V&V) effectiveness or efficiency, and productivity. The IEEE Standard 1012 for Software Verification and Validation [25] explains the first three as follows: *Defect density* measures can provide insightful information concerning the software product quality, the quality of the software development processes, and the quality of the V&V effort to discover anomalies in the system/software and to facilitate correction of the anomalies. Measures associated with *V&V effort effectiveness* provide quantitative indications that characterize the added benefits of V&V to discover anomalies in software products and processes. Measures associated with *V&V effort efficiency* provide data that characterize the capability of the V&V effort to discover anomalies in software products and processes in the development activity into which they are injected. *Productivity*, on the other hand, measures the performance of the production. It is a ratio of production output to what is required to produce it – inputs. Although its usefulness in software engineering is arguable, it is simply calculated by the ratio of product size to production effort [24].

The International Standard "ISO/IEC 15939 Systems and software engineering — Measurement process" [9] explains the process and outcomes of measuring systems and software in a systematic way. It defines the purpose of the measurement process as "to collect, analyze, and report data relating to the products developed and processes implemented within the organizational unit, to support effective management of the processes, and to objectively demonstrate the quality of the products". According to this standard, as a result of successful implementation of the measurement process, the following outputs are achieved: Organizational commitment for measurement is established and sustained; the information needs of technical and management processes are identified; an appropriate set of measures, driven by the information needs are identified and/or developed; measurement activities are identified; identified measurement activities are planned; the required data is collected, stored, analyzed, and the results interpreted; information products are used to support decisions and provide an objective basis for communication; the measurement process and measures are evaluated; and improvements are communicated to the measurement process owner.

The Goal Question Metric (GQM) Approach [8] proposes that measurement definition must be top-down as based on goals and models. The approach has a hierarchical structure. It starts with a goal specifying purpose of the measurement, object to be measured, issue to be measured, and viewpoint from which the measure is taken. Objects of measurement include products, processes, and resources. The goal is refined into several questions that usually break down the issue into its major components. Questions try to characterize the object of measurement (product, process, or resource) with respect to a selected quality issue, and to determine its quality from the selected viewpoint. Each question is then refined into measures, either objective or subjective. The same measure can be used to answer different questions under the same goal. The result of the application of the GQM Approach is the specification of a measurement system targeting a particular set of issues and a set of rules for the interpretation of the measurement data. The approach can be used in isolation or within the context of a more general approach to software quality improvement.

2.3. Capability Maturity Model Integration (CMMI)

CMMI is a process reference model that was developed by the Software Engineering Institute (SEI) of Carnegie Mellon University. It addresses the development and maintenance activities applied to both products and services. This model could be used for improving processes, and measuring the capability of a process or the maturity of an organization [10]. CMMI components (including a model, its training materials, and appraisal related documents) are designed to meet the needs of some specific areas of interest, which is called constellation. Three constellations are CMMI for Development, CMMI for Services, and CMMI for Acquisition.

CMMI consists of process areas where domain related process activities are explained, goals and practices of these process areas, and two different representations which are staged and continuous. The representations can be considered as two different viewpoints created by putting the model components together in two different ways and indicate in what way the goals and practices shall be handled. Model components are grouped into three categories which are required, expected, and informative. The satisfaction of required and expected components are questioned formally in process assessments.

A process area is a cluster of related practices in an area that, when implemented collectively, satisfy a set of goals considered important for making improvement in that area. In CMMI for Development [10], all system and software development processes are addressed by 22 process areas such as Project Planning, Requirements Development, Measurement and Analysis, and these process areas are the same for both staged and continuous representations. Process areas are organized by maturity levels (ML1–ML5) to identify organizational maturity in the staged representation and rated by capability levels (CL0–CL5) to identify process capability in the continuous representation.

2.4. An Assessment Approach for Quantitative Process Management (A^2QPM)

An Assessment Approach for Quantitative Process Management (A^2QPM) aims to increase the use of quantitative management in software engineering even when organizations lack formal systems for process measurement [11]. It facilitates the evaluation of a process and measures for quantitative analysis from process and measurement perspectives. For the former it provides assets to perform systematic clustering of process executions and data. For the latter it offers assets to investigate measurement practices, and measure and data usability for quantitative analysis. The context and perspectives of the A^2QPM is shown in Fig. 1 [26].

Quantitative techniques such as *statistical process control (SPC)* have been utilized in manufacturing disciplines since the 1920s to understand process variation and improve process performance. Variation is inherent to a process; however, while some processes

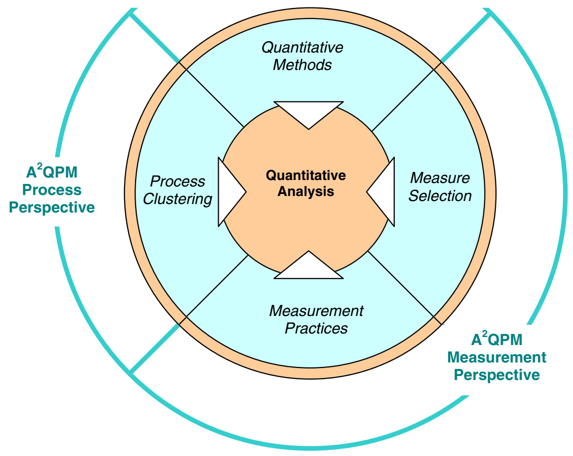


Fig. 1. Assessment perspectives of the A^2QPM [26].

display controlled variation, others display uncontrolled variation [27]. Controlled variation means stable and consistent variation over time and consists of *common causes*. Uncontrolled variation, on the other hand, indicates a pattern that changes over time in an unpredictable manner and is characterized by *special causes*. For statistical process control be possible, it is required that process data come from a single and constant system of causes (or common causes) [28].

Software process data often represent multiple sources that need to be treated separately and discovering these multiple sources requires the careful investigation of process executions. Stratification is a technique used to analyze or divide a universe of data into homogeneous groups [27]. The A^2QPM aims to cluster process executions as stemming from a single and constant system of causes in accordance to the changes in the values of process attributes such as inputs, outputs, activities, roles, tools and techniques. If the executions of a process show similarity in terms of these attributes, then process executions form a homogeneous subgroup (or "cluster") that consistently performs within itself. The process cluster, then, is a candidate for statistical management.

Measurement perspective includes the amplification of basic measurement practices as well as the existence and characteristics of the measurement data. Even if no measurement process is defined, one can investigate the practices applied during data collection and analysis. The A^2QPM identifies a number of usability attributes (MUAs) such as measure identity, data existence, data verifiability, data dependability, data normalizability and data integrability. The approach provides a questionnaire based on these attributes, and recommends investigating the usability of a measure for quantitative analysis prior to implementation. Metric identity (MUA-1) deals with basic characteristics of a measure such as entity and attribute, scale type, unit, formula, data type and range. Data existence (MUA-2) addresses the organization and availability of measurement data points. Data verifiability (MUA-3) is related to the consistency in the recording and storing of measurement data in executions. Data dependability (MUA-4) requires all measurement data be recorded as close as possible to its source with accuracy and precision. The awareness of data collectors of the issues related to measurement data (why it is collected, how it is utilized, etc.) plays a significant role in data dependability. Data normalizability and data integrability are related to the usefulness of a measure and should be satisfied if the quantitative analysis is to provide more insight into process understanding and improvement. These two attributes are informative and not counted in usability evaluation.

The steps of the assessment process are not given here due to space constraint and interested readers can refer to [26] for a detailed description. However, the forms defined by the A^2QPM and used in the process are briefly explained below for refering later in this paper.

Process Execution Record: This form captures the instant values of internal process attributes for a process execution. The actual values of inputs, outputs, activities, roles, tools and techniques for a specific process execution are recorded on the form. Recorded values are used to identify the merged list of process attribute values which are shown in the Process Similarity Matrix for verification.

Process Similarity Matrix (PSM): The matrix is created based on the values previously entered into Process Execution Records to verify the values of internal process attributes against process executions. The values of internal process attributes are shown in the rows, and process execution numbers are shown in the columns of the matrix. By going over the executions, the values of internal process attributes are questioned and marked if applicable for each process execution. The completed matrix helps us to see the differences between process executions in terms of process attribute values and enables us to identify the clusters of process executions accordingly.

Process Execution Questionnaire: This records the assignable causes for a process execution in terms of outer factors such as changes in process performers and process environments. While working retrospectively on existing process data, a questionnaire is completed for every out-of-control point on a control-chart and answers are used to understand the assignable causes.

Measure Usability Questionnaire (MUQ): This is used to investigate the usability of a process measure in quantitative analysis in terms of measure usability attributes. It includes a number of questions and rules for rating the usability attributes. There are two types of questionnaire for base and derived measures separately. The template of the MUQ proposed for base measures is shown in Fig. 2. Expected answers, which are given in the rightmost column in the figure, represent the desired answers to the questions targeted to evaluate the attributes in the leftmost column. Answers to some questions are not used in the evaluation and expected answers to such questions are marked with an "–". The color-coding in the "Rating" column can also be used to follow the questions, answers of which are used as a base for the evaluation. The questions at the rows of the white cells in the "Rating" column are targeted to evaluate the usability of the measure in quantitative analysis.

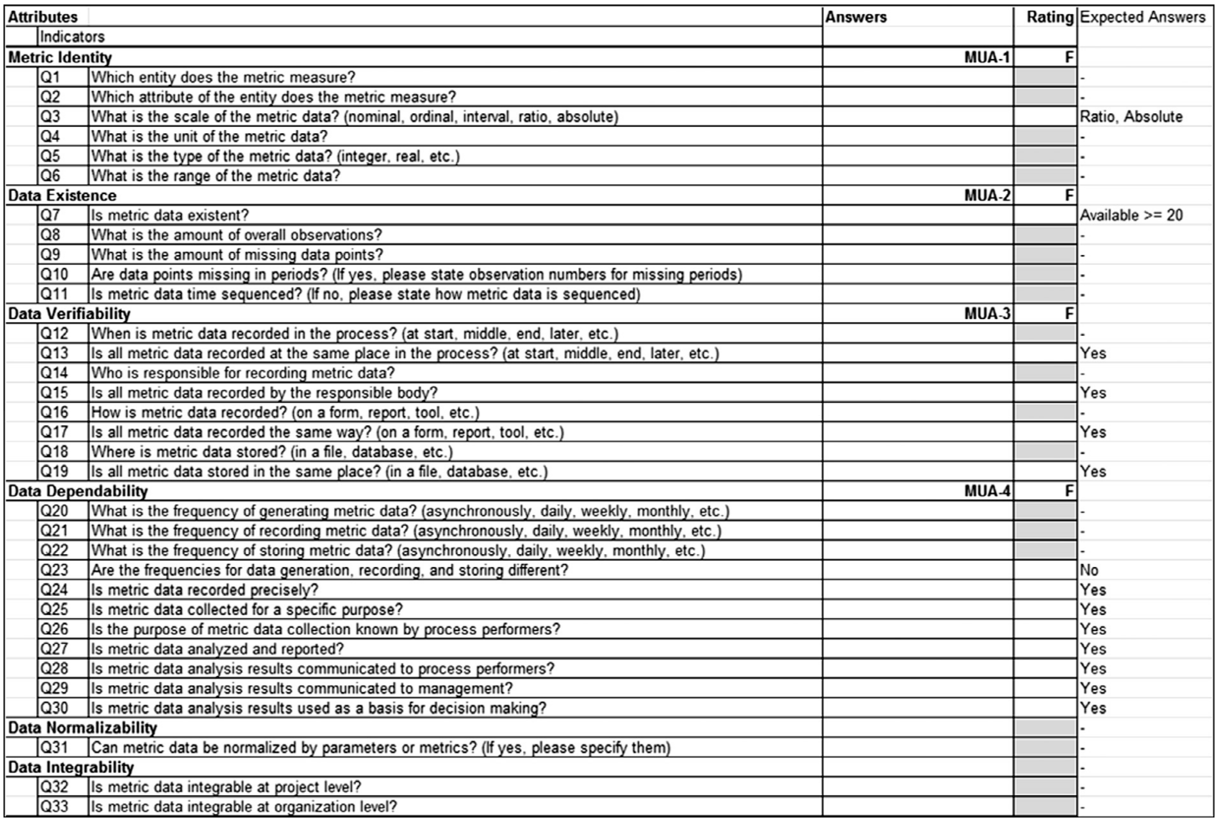
The degree of satisfaction of a measure usability attribute is rated in four values in the ordinal scale (N representing the poorest satisfaction degree): None (N), Partial (P), Large (L), and Fully (F). MUA-1 (metric identity attribute) is rated as either N or F, and satisfied if the scale of the measure is Ratio or Absolute. MUA-2 (data existence attribute) is also rated as either N or F, and satisfied if the number of measurement data points is at least 20. MUA-3 (data verifiability attribute) is rated based on the number of questions which have expected answers. MUA-3 is rated as N, P, L, F, if the number of questions answered as expected is 1, 2, 3, or 4, respectively. MUA-4 (data dependability attribute) is rated as similar to MUA-3. MUA-4 is rated as N, P, F, or L, if the minimum number of questions answered as expected is 1, 3, 5, or 7, respectively.

The usability of a measure is decided based on the ratings of the measure usability attributes, and can take the following four values in the ordinal scale (N representing the poorest usability): Strong (S), Moderate (M), Weak (W), and None (N). The sequence of the ratings for the attributes determines the result of the evaluation. If the sequence of the ratings for the attributes 1 through 4 is FFFF or FFFL, then the usability is rated as Strong. If the sequence is FFFP, FFLL, or FFLP, the usability is rated as Moderate. If the sequence is FFFN, FFLN, FFPP, FFPN, or FFNN, then the usability is rated as Weak. In the other cases, the usability is rated as None. We recommend including measures with a usability rating of Strong or Moderate in the quantitative analysis.

Among the assets described above, PSM and MUQ were used in the analysis method which is described in Section 4 and was followed to evaluate the performances of the Incremental and Agile processes. PSMs were completed for the phases of the development processes and provided qualitative data on the components and enactments of the processes compared. This data was then utilized in deriving the rationale behind the quantitative results. MUQs were completed for each base measure included in the analyses and comparison, and constituted a valuable asset for both the operational definition of the measure and its evaluation of usability in the quantitative analysis.

2.5. Related work

Dyba and Dingsøyr [4] performed a systematic literature review on empirical studies of Agile software development which were reported until and including the year 2005, and identified 36



Attributes			Answers	Rating	Expected Answers
	Indicators				
Metric Identity			MUA-1	F	
	Q1	Which entity does the metric measure?			-
	Q2	Which attribute of the entity does the metric measure?			-
	Q3	What is the scale of the metric data? (nominal, ordinal, interval, ratio, absolute)			Ratio, Absolute
	Q4	What is the unit of the metric data?			-
	Q5	What is the type of the metric data? (integer, real, etc.)			-
	Q6	What is the range of the metric data?			-
Data Existence			MUA-2	F	
	Q7	Is metric data existent?			Available >= 20
	Q8	What is the amount of overall observations?			-
	Q9	What is the amount of missing data points?			-
	Q10	Are data points missing in periods? (If yes, please state observation numbers for missing periods)			-
	Q11	Is metric data time sequenced? (If no, please state how metric data is sequenced)			-
Data Verifiability			MUA-3	F	
	Q12	When is metric data recorded in the process? (at start, middle, end, later, etc.)			-
	Q13	Is all metric data recorded at the same place in the process? (at start, middle, end, later, etc.)			Yes
	Q14	Who is responsible for recording metric data?			-
	Q15	Is all metric data recorded by the responsible body?			Yes
	Q16	How is metric data recorded? (on a form, report, tool, etc.)			-
	Q17	Is all metric data recorded the same way? (on a form, report, tool, etc.)			Yes
	Q18	Where is metric data stored? (in a file, database, etc.)			-
	Q19	Is all metric data stored in the same place? (in a file, database, etc.)			Yes
Data Dependability			MUA-4	F	
	Q20	What is the frequency of generating metric data? (asynchronously, daily, weekly, monthly, etc.)			-
	Q21	What is the frequency of recording metric data? (asynchronously, daily, weekly, monthly, etc.)			-
	Q22	What is the frequency of storing metric data? (asynchronously, daily, weekly, monthly, etc.)			-
	Q23	Are the frequencies for data generation, recording, and storing different?			No
	Q24	Is metric data recorded precisely?			Yes
	Q25	Is metric data collected for a specific purpose?			Yes
	Q26	Is the purpose of metric data collection known by process performers?			Yes
	Q27	Is metric data analyzed and reported?			Yes
	Q28	Is metric data analysis results communicated to process performers?			Yes
	Q29	Is metric data analysis results communicated to management?			Yes
	Q30	Is metric data analysis results used as a basis for decision making?			Yes
Data Normalizability					-
	Q31	Can metric data be normalized by parameters or metrics? (If yes, please specify them)			-
Data Integrability					-
	Q32	Is metric data integrable at project level?			-
	Q33	Is metric data integrable at organization level?			-

Fig. 2. The template of Measure Usability Questionnaire for base measures.

research studies which were of acceptable rigour, credibility, and relevance. The authors categorized these studies into four thematic groups, namely introduction and adoption, human and social factors, perceptions of Agile methods, and comparative studies; and identified a number of reported benefits and limitations of Agile development within each of these themes. The authors reported that one third (11) of the reviewed primary studies provided some form of comparison of Agile development against an alternative.

A number of relevant studies was reported after the year 2005, and most of them evaluate Agile processes and their mostly used models XP and Scrum. Khramov [29] performed a study to determine the cases where XP practices may be counter-productive while producing quality code. Benefield [30] demonstrated the effect of adopting Scrum in Yahoo! development teams. Ji and Sedano [31] compared the effect of transition from traditional development to Agile development by using sizes of work products.

A list of comparative studies utilizing a control group and following a methodological approach, which includes the ones reported by Dyba and Dingsøyr [4] until the year 2006 and the ones published thereafter as surveyed by the authors of this paper, is given in Table 2.

The main contributions of this paper are the combined utilization of qualitative and quantitative methods, and detailed description of the methodological steps applied in the comparison. From the studies listed in Table 2, only Petersen and Wohlin [5] provided

Table 2
Study aims for comparative studies.

Year [Ref.]	Study Ref in [4]	Study aim
2003 [32]	S15	Compare XP with a traditional approach with respect to quality
2004 [33]	S6	Compare the effectiveness of incremental and evolutionary process models for technology development projects
2004 [34]	S10	Compare empirical findings of an Agile method based on XP with a baseline, with respect to productivity, defect rates, cost, and schedule deviations
2004 [35]	S14	Compare the effectiveness of XP with traditional development, with respect to pre- and post-release quality, programmer productivity, customer satisfaction, and team moral
2005 [36]	S7	Compare differences in resource utilization and efficiency in products developed using sequential, incremental, evolutionary, and extreme programming
2005 [37]	S28	Compare Agile and document-driven approaches in managing uncertainty in software development
2005 [38]	S32	Compare plan-driven and Agile development with respect to team cohesion and product quality
2006 [39]	–	Compare the return of investment of Agile and traditional practices of undergraduate teams
2007 [40]	–	Compare a combination of Agile methods and CMMI with traditional development
2010 [5]	–	Reveal the effect of moving from a plan-driven to an incremental and Agile development approach
2010 [41]	–	Compare the performances of two projects adopting Scrum and using traditional development

a detailed description of their empirical method and used process performance measurements to support or contradict their qualitative findings.

In this study, qualitative data regarding the processes was represented by process modeling and elicited via Process Similarity Matrices to understand process components, and identify any deficiency in the definition or inconsistency in the enactment. Qualitative data regarding the measures was elicited via Measure Usability Questionnaires to decide measures' usability in the analysis. Quantitative data of the measures was gathered retrospectively from the tools and the forms. Quantitative results obtained at the end of the comparison were elaborated in the light of the qualitative data synthesized for the processes.

3. Case study outline

3.1. Context and scope

The case study was implemented in a middle-size, telecommunication software development company that employed 65 developers and 25 quality assurance staff. The number of projects run simultaneously was five and the number of software product families maintained was three at the time of conducting the study.

The projects A and B, in which the Incremental and Agile processes were employed, have been carried out to create one of the product families, and the development of 20 features in each project were taken as the base for the analyses and comparison. At the time of performing the analysis in the project B, the development of the product features by applying the Agile Process was in progress; and, the development of the 20 features were included in the analysis. Therefore, the features from the projects could not be selected as the same due to the difference in development priority of the features in the project B. The length of the selected features varied from 140 to 610 lines of code with an average value of 318 in the project A, and from 95 to 690 lines of code with an average value of 339 in project B. The features selected were not identical, but the domain, the infrastructure, the programming language and environment, and the development team were similar in the projects. The project A consisted of a project manager, a project leader, four developers, a tester, a customer support staff, a technical service staff, and a configuration manager (a total of 10 people) whereas the project B consisted of; a project manager, a project leader, five developers, a tester and a configuration manager (a total of 9 people). The number of persons employed in common in the projects was 7.

The Incremental Process employed in the project A is an adaption of the Waterfall Model by the organization. The software product subject to analyses in this study had been previously developed with the Incremental Process, and had a great interest and customer satisfaction in the inner and outer markets at the beginning. However, as the requirements of revision and extension were emerged in time, the complexity of the product was increased. Major defects that had been introduced and could not be caught through the increments were later reported by the customers. This led to the loss of the customers and increased development costs. Due to these problems and the marketing strategy, the company decided on the development of the product with an extended scope and by applying an Agile Process. The newly introduced Agile Process is a combination of the Unified Software Development Process, Extreme Programming, and Scrum, adapted to the specific needs of the project B. Neither the Incremental and Agile processes nor the measurement process had been defined organization-wide. However, many projects had been following the Incremental Process on a plan-driven basis.

For both the Incremental and Agile processes; the phases of Implementation (design and coding), System Test, and Customer Use were identified as the base for the analyses and comparison. For comparing the scope of the development processes, CMMI for Development [10] was taken as the process reference model since the organization had a target to achieve CMMI maturity level 3. The steps of the Incremental and Agile processes were mapped to the practices of the CMMI process areas, namely Requirements Management, Process and Product Quality Assurance, Requirements Development, Technical Solution, Product Integration, Validation, and Verification. The process areas regarding project management were not included in the study since our concern was to understand the performances of the development and testing practices. The way the mapping was performed is not detailed here and explained in Section 4. In order to summarize the matching of the scopes of the Incremental and Agile processes with respect to the specific goals of the selected CMMI process areas, Table 3 shows the degrees of achievement in the ordinal scale (None, Partial, Large, Full). As a result of the comparison of the scopes, gradation differences were observed between the processes, especially in Requirements Management, Technical Solution, and Validation process areas. It was detected that the Incremental Process had less achievement of the goals of these process areas than those of the Agile Process. There were also some differences in the achievement of the goals of Process and Product Quality Assurance and Product Integration process areas. However, since the degree of achievement of no specific goal of no process area was graded as "None", it was concluded that the scopes of the Incremental and Agile processes generally match each other in terms of the selected process areas of the CMMI for Development.

3.2. Objective and design

Our purpose was to compare the performance of the plan-driven, the Incremental Process with that of the Agile Process of a middle-size software company. To satisfy this purpose, we identified the following research question (RQ):

RQ: How do Incremental Process and Agile Process compare to each other in terms of process performance and product quality?

The analyses of the two processes as the base for the comparison were designed as a multiple-case and embedded study [42] as a whole. The structure of the design is shown in Fig. 3. While structuring the design, we considered the existence of data for the sub-units of analyses, and the accessibility of process performers. At least 20 measurement data items are required [43], and interviews with process performers are necessary to assess consistency of process executions and usability of measures [11].

The context of the analyses was a middle-size software company, and the units of analyses were the Incremental Process and the Agile Process. These processes were applied in two different projects A and B, but for developing the same product X in the telecommunications domain. The sub-units of analyses are in pairs of three types: Product-Measure, Process Phase-Measure, and Overall Process-Measure. In other words, the performances of the development processes were evaluated and compared in terms of product, process, and process-phase measures.

3.3. Analysis goals and measures

The GQM Approach was applied in a top-down fashion in order to determine the goals and measures for the analyses of the Incremental and Agile processes. The analysis goals were identified, the questions for each goal were derived, and one or more measure(s) were determined to answer these questions. Since the purpose of the study was to compare the Incremental and Agile processes in

Table 3
The scopes of the Incremental and Agile processes with respect to the specific goals of the selected CMMI process areas.

Process area	Maturity level	Specific goal (SG)	Degree of achievement	
			Incremental	Agile
Requirements Management (REQM)	2	SG 1 Manage Requirements	Partial	Full
Processes and Products Quality Assurance (PPQA)	2	SG 1 Objectively Evaluate Processes and Work Products	Large	Large
		SG 2 Provide Objective Insight	Large	Large/Full
Requirements Development (RD)	3	SG 1 Develop Customer Requirements	Full	Full
		SG 2 Develop Product Requirements	Full	Full
		SG 3 Analyze and Validate Requirements	Large	Large
Technical Solution (TS)	3	SG 1 Select Product Component Solutions	Partial	Large
		SG2 Develop the Design	Large	Large
		SG 3 Implement the Product Design	Large	Large/Full
Product Integration (PI)	3	SG 1 Prepare for Product Integration	Large	Large
		SG 2 Ensure Interface Compatibility	Partial/Large	Large
		SG 3 Assemble Product Components and Deliver the Product	Large	Large
Validation (VAL)	3	SG 1 Prepare for Validation	Large	Large
		SG 2 Validate Product or Product Components	Large	Full
Verification (VER)	3	SG 1 Prepare for Verification	Full	Full
		SG 2 Perform Peer Reviews	Large	Large
		SG 3 Verify Selected Work Products	Large/Full	Large/Full

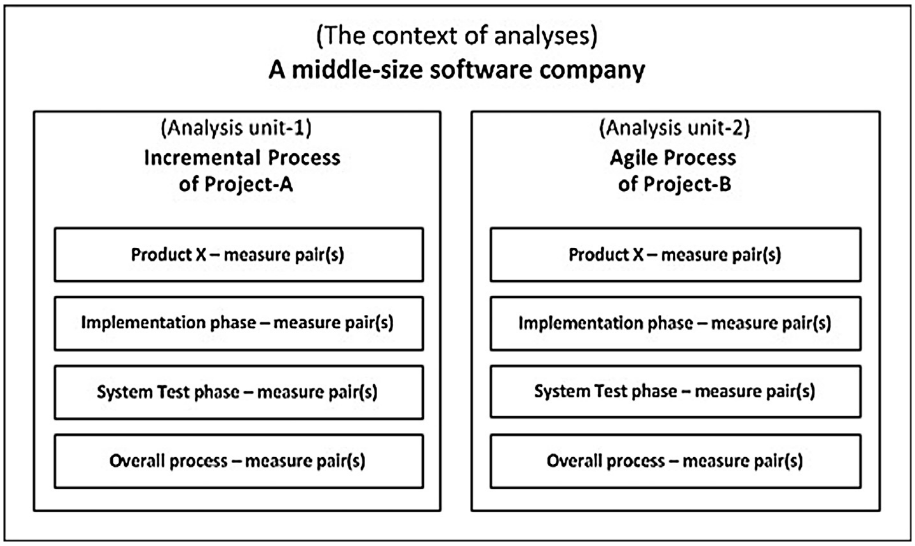

(The context of analyses)
A middle-size software company
(Analysis unit-1)
Incremental Process of Project-A
Product X – measure pair(s)
Implementation phase – measure pair(s)
System Test phase – measure pair(s)
Overall process – measure pair(s)
(Analysis unit-2)
Agile Process of Project-B
Product X – measure pair(s)
Implementation phase – measure pair(s)
System Test phase – measure pair(s)
Overall process – measure pair(s)

Fig. 3. Analysis design.

accordance with the determined goals, there had been no differentiation on the basis of the questions that corresponded to the goals in the GQM tables and the derived measures which answered these questions. The GQM table, essential for the comparison, is given in Table 4. The definitions of the derived measures (D1–D19) are given in Table 5 by generalizing their names.

Although there was no differentiation in the derived measures in the GQM tables created for the analyses of the Incremental and Agile processes, there were some differences in the base measures which formed the derived measures. Base measures, which were essential to analyses, and their names in both of the development processes are given in Table 6. For instance, in the "Defect Count" category, "Number of defects detected in System Test" in the Incremental Process corresponds "Number of defects detected in System Test" and "Number of defects detected in Automation Test" in the Agile Process.

Data belonging to the base measures defined for the processes was collected by tools and forms, and verified by holding interviews with data providers and by cross-checking with produced documents where available. The tools used during data collection are Defect Tracking System, Test Management Tool, and Configuration Management Tool. The forms used during data collection were daily and weekly created Microsoft Excel sheets.

Data elicitation, gathering, and analysis were performed by the second author of this study who was a member of the quality assurance staff in both developments. Internal review of the data gathered and analyzed was done in two meetings of the quality assurance team. External review of the data and analyses was performed by the first author of the study.

Understanding the attributes of the base measures defined for the processes and the availability of them for the quantitative analyses were significant in the inclusion of these measures into the analyses. To achieve such an understanding, Measure Usability Questionnaire described in Section 2.4 was used for each base measure defined in Table 6 in order to assure its usability in the quantitative analysis. The attributes of Data Verifiability (MUA-3) and Data Dependability (MUA-4) were rated as Large and Partial, respectively, for some of these measures. They were software

Table 4
The GQM table used for comparison of the Incremental and Agile processes.

Goal	**G1**	**To understand product quality**
Question	*Q1*	*What is test defect density in Implementation phase?*
Measure	D1	Test defect density per test scenario in Implementation phase
	D2	Test defect density per line of code in Implementation phase
Question	*Q2*	*What is test defect density in System Test phase?*
Measure	D3	Test defect density per test scenario in System Test phase
	D4	Test defect density per line of code in System Test phase
Question	*Q3*	*What is test defect density in Customer phase?*
Measure	D5	Test defect density in Customer phase
Question	*Q4*	*What is test defect density in overall process?*
Measure	D6	Overall test defect density
Goal	**G2**	**To understand performance of System Test phase**
Question	*Q5*	*What is test execution verification and validation (V&V) effectiveness in System Test phase?*
Measure	D7	Test execution V&V effectiveness in System Test phase
Question	*Q6*	*What is test effectiveness in System Test phase?*
Measure	D8	Test effectiveness in System Test phase
Question	*Q7*	*What is test speed in System Test phase?*
Measure	D9	Test speed in System Test phase
Question	*Q8*	*What is effort estimation capability in System Test phase?*
Measure	D10	Effort estimation capability in System Test phase
Goal	**G3**	**To understand performance of Implementation phase**
Question	*Q9*	*What is productivity in Implementation phase?*
Measure	D11	Productivity in Implementation phase
Question	*Q10*	*What is effort estimation capability in Implementation phase?*
Measure	D12	Effort estimation capability in Implementation phase
Question	*Q11*	*What is test execution V&V effectiveness in Implementation phase?*
Measure	D13	Test execution V&V effectiveness in Implementation phase
Question	*Q12*	*What is test effectiveness in Implementation phase?*
Measure	D14	Test effectiveness in Implementation phase
Question	*Q13*	*What is test speed in Implementation phase?*
Measure	D15	Test speed in Implementation phase
Goal	**G4**	**To understand overall process performance**
Question	*Q14*	*What is test execution V&V effectiveness in overall process?*
Measure	D16	Test execution V&V effectiveness in overall process
Question	*Q15*	*What is defect removal effort ratio in overall process?*
Measure	D17	Defect removal effort ratio in overall process
Question	*Q16*	*What is productivity in overall process?*
Measure	D18	Productivity in overall process
Question	*Q17*	*What is effort estimation capability in overall process?*
Measure	D19	Effort estimation capability in overall process

Table 5
Description of derived measures.

Derived measure	Unit	Textual description
Test Defect Density (per Test Scenario)	No. of defects/No. of test scenarios	In a specific process phase, it is the number of defects found in software per number of test scenarios executed
Test Defect Density (per Line of Code)	No. of defects/Lines of code	In a specific process phase or in overall process, it is the number of defects found in software per lines of code developed
Test Execution V&V Effectiveness	No. of defects/No. of defects	In a specific process phase or in overall process, it is the ratio of number of defects found to the number of all defects detected (including the ones reported by the customer) in software
Test Effectiveness	No. of defects/Person-hours	In a specific process phase, it is the number of defects found in software per unit of effort spent for test design and execution
Test Speed	No. of test scenarios/Person-hours	In a specific process phase, it is the number of test scenarios executed per unit of effort spent for test design and execution
Effort Estimation Capability	Person-hours/Person-hours	In a specific process phase or in overall process, it is the ratio of actual effort spent to the planned effort
Productivity	Lines of code/Person-hours	In a specific process phase or in overall process, it is the lines of code developed per unit of actual effort spent
Defect Removal Effort Ratio	Person-hours/Person-hours	In overall process, it is the ratio of actual effort spent for removing detected defects to the overall actual effort spent

Table 6
The mapping of base measures for the Incremental and Agile processes.

Measure category	Base measure in the Incremental Process	Base measure in the Agile Process
Defect Count	Number of defects detected in Implementation	Number of defects detected in Unit Test Number of defects detected in Integration Test Number of defects detected in Smoke Test
	Number of defects detected in System Test	Number of defects detected in System Test Number of defects detected in Automation Test
	Number of defects detected in Customer	Number of defects detected in Customer
Test Scenario Count	Number of test scenarios run by the developer	Number of unit test scenarios run by the developer Number of integration test scenarios run by the developer Number of smoke test scenarios run by the developer
	Number of test scenarios run by the tester	Number of system test scenarios run by the tester Number of automation test scenarios run by the tester
Implementation Effort	Software design effort Software coding effort Test design effort of the developer Test execution effort of the developer	Software design effort Software coding effort Test design effort of the developer Test execution effort of the developer
System test effort	Test design effort of the tester Test execution effort of the tester Defect reporting effort Re-test effort for detected defects	Test design effort of the tester Test execution effort of the tester Defect reporting effort Re-test effort for detected defects
Defect removal effort	Removal effort of defects detected in Implementation Removal effort of defects detected in System Test Removal effort of defects detected in Customer	Removal effort of defects detected in Implementation Removal effort of defects detected in System Test Removal effort of defects detected in Customer
Estimated effort	Planned effort for Implementation Planned effort for System Test	Estimated effort for Implementation Estimated effort for System Test
Software length	Source lines of code	Source lines of code

design effort, software coding effort, test design effort of the developer, and test execution effort of the developer. Since Data Dependability of these four measures was evaluated as Partial by the questionnaires, their data was verified through the interviews with the primary sources. When the GQM tables were still draft plans, two derived measures called Planned Duration and Actual Duration had been defined. However, since their usability evaluated by the questionaries was weak, they were excluded from the analyses and comparison.

4. Analysis method

The analyses of the Incremental and Agile processes were carried out by following the steps of an analysis method shown in Fig. 4. The method was defined with the purpose of providing a systematic, repeatable roadmap to analyze a software development process. It can be used to understand the performance of a single process or a group of processes for the purpose of comparison.

In the following paragraphs, we describe analysis steps in detail and provide examples for their outputs from the evaluation of the Incremental Process [44].

The first step is identifying basic phases of the software development process. The phases may include requirements analysis, architectural design, detailed design, implementation, integration, system test, and etc. This is a preparation step to understand the context of the software development.

For the Incremental Process; the phases of Development (design and implementation), System Test, and Customer Use were identified for the quantitative analysis.

The steps 2 and 3 are performed for each software development phase identified in step 1. In the step 2, the flow of a software

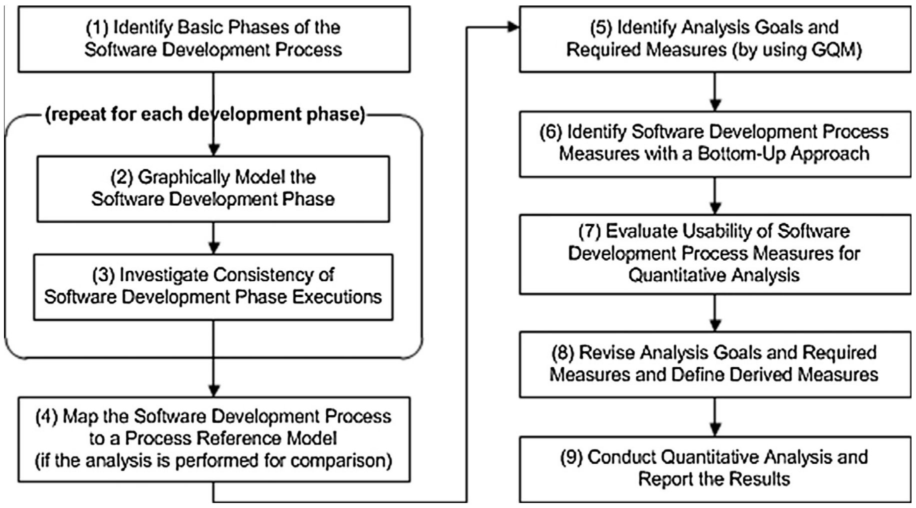

(1) Identify Basic Phases of the Software Development Process
(repeat for each development phase)
(2) Graphically Model the Software Development Phase
(3) Investigate Consistency of Software Development Phase Executions
(4) Map the Software Development Process to a Process Reference Model (if the analysis is performed for comparison)
(5) Identify Analysis Goals and Required Measures (by using GQM)
(6) Identify Software Development Process Measures with a Bottom-Up Approach
(7) Evaluate Usability of Software Development Process Measures for Quantitative Analysis
(8) Revise Analysis Goals and Required Measures and Define Derived Measures
(9) Conduct Quantitative Analysis and Report the Results

Fig. 4. Analysis method.

development phase is modelled graphically to understand its basic attributes such as inputs, activities, outputs, roles, and tools. If there is an organization-wide definition of the software development process, it can be used as an input to this step. Otherwise, project plans and interviews with project managers and process performers can be used to capture phase details. We recommend

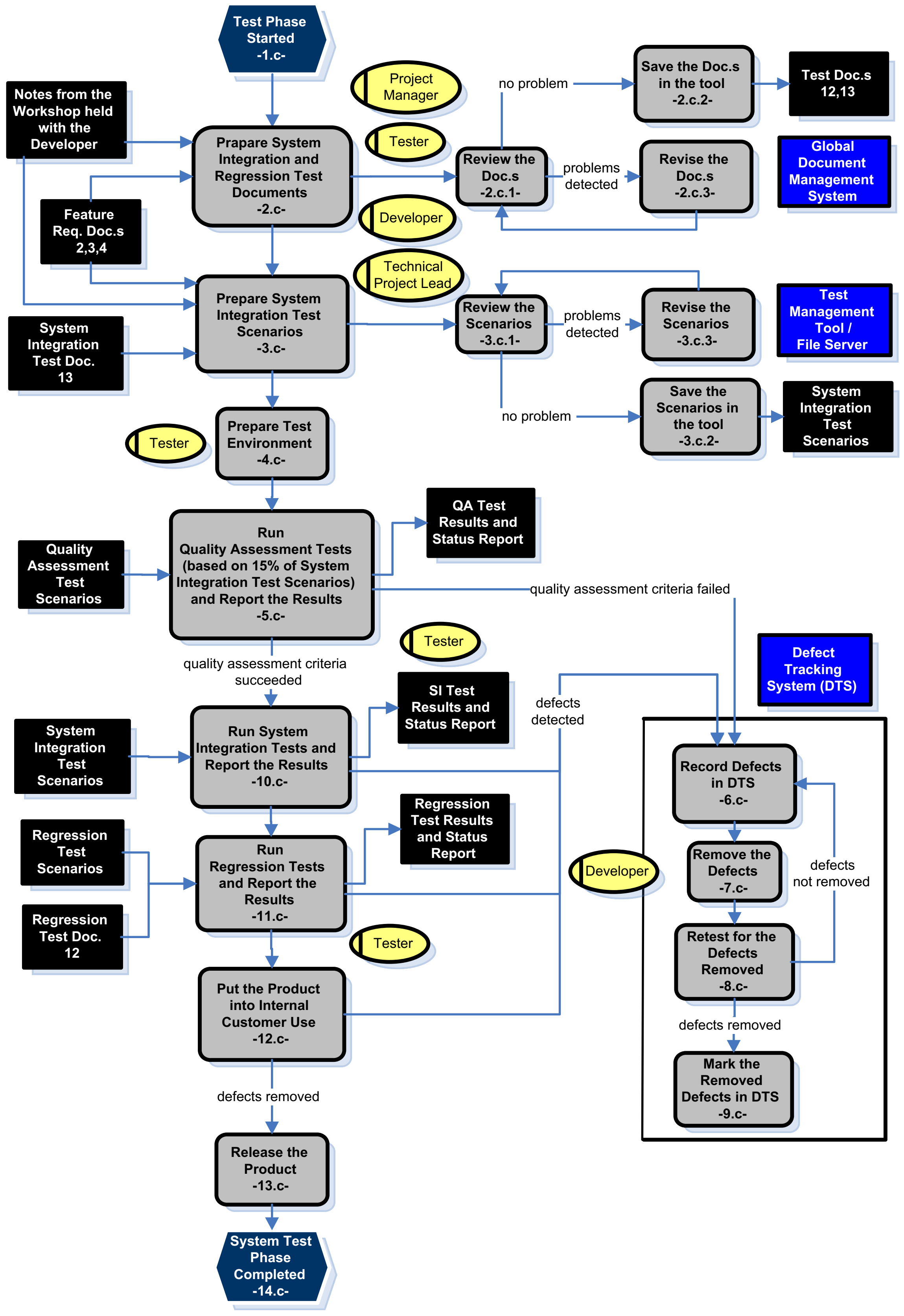

Test Phase Started -1.c-
Notes from the Workshop held with the Developer
Project Manager
Tester
Developer
Technical Project Lead
Prapare System Integration and Regression Test Documents -2.c-
Feature Req. Doc.s 2,3,4
no problem
Save the Doc.s in the tool -2.c.2-
Test Doc.s 12,13
Review the Doc.s -2.c.1-
problems detected
Revise the Doc.s -2.c.3-
Global Document Management System
System Integration Test Doc. 13
Prepare System Integration Test Scenarios -3.c-
Review the Scenarios -3.c.1-
problems detected
Revise the Scenarios -3.c.3-
Test Management Tool / File Server
no problem
Save the Scenarios in the tool -3.c.2-
System Integration Test Scenarios
Tester
Prepare Test Environment -4.c-
QA Test Results and Status Report
Quality Assessment Test Scenarios
Run Quality Assessment Tests (based on 15% of System Integration Test Scenarios) and Report the Results -5.c-
quality assessment criteria failed
Tester
quality assessment criteria succeeded
Defect Tracking System (DTS)
SI Test Results and Status Report
defects detected
System Integration Test Scenarios
Run System Integration Tests and Report the Results -10.c-
Record Defects in DTS -6.c-
Regression Test Scenarios
Regression Test Results and Status Report
Run Regression Tests and Report the Results -11.c-
Developer
Remove the Defects -7.c-
defects not removed
Regression Test Doc. 12
Tester
Retest for the Defects Removed -8.c-
Put the Product into Internal Customer Use -12.c-
defects removed
Mark the Removed Defects in DTS -9.c-
defects removed
Release the Product -13.c-
System Test Phase Completed -14.c-

Fig. 5. Process model of System Test phase of the Incremental Process [44].

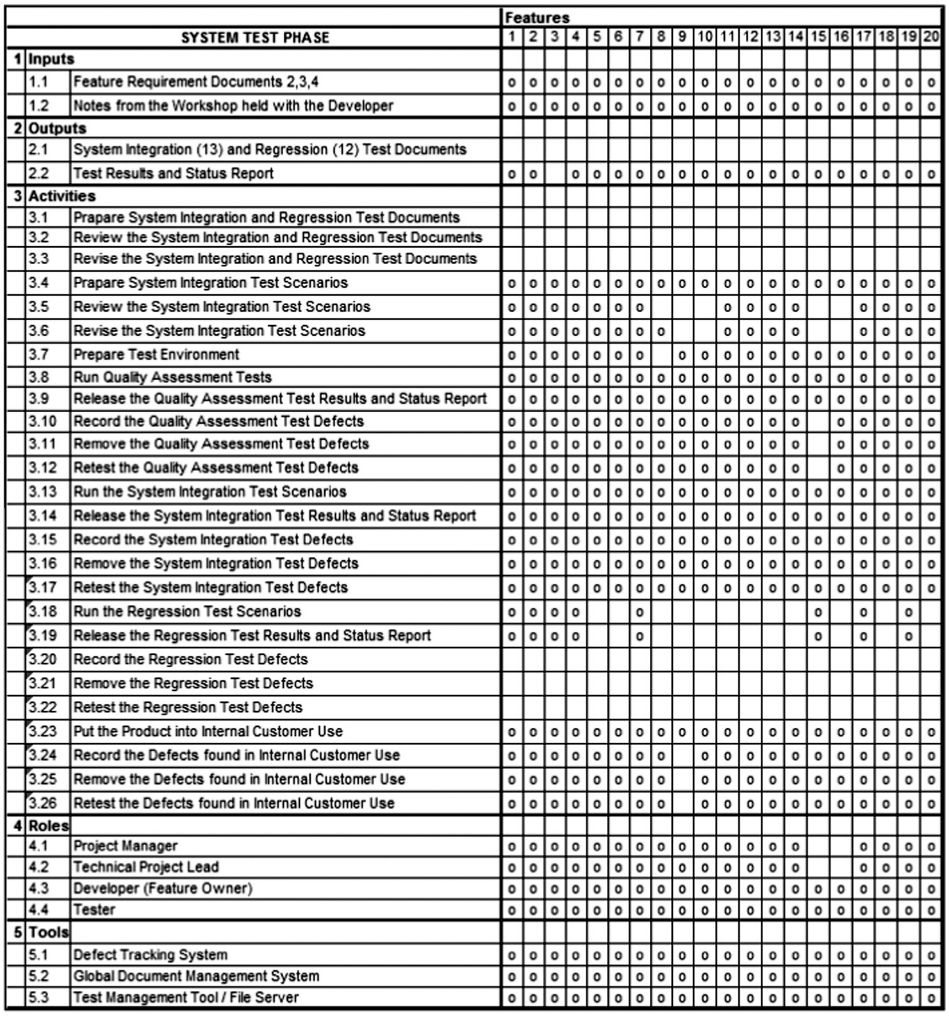
using Event-Driven Process Chain (EPC) notation [45] to graphically model a phase, but any other notation to capture process flow and phase attributes specified above may serve the same purpose. In the model, each activity of the phase is numbered for a later reference in the step 4. The phase attributes elicited by the process model constitute the process components in definition. These attributes are then entered (as each to appear once) into the rows of the Process Similarity Matrix to form a base for their verification against process enactments in the step 3.

The EPC model of the System Test phase of the Incremental Process is shown in Fig. 5.

In the step 3, the consistency of software development phase executions are verified against phase attributes captured in the previous step. If the attributes are similar between phase executions, it means the phase is consistently performed. This provides the confidence that phase measurement data come from a homogeneous source and can be utilized for quantitative analysis. The Process Similarity Matrix (PSM) described in Section 2.4 is utilized to capture the values of process attributes in process executions. Process records and interviews with process performers are used as the information sources in verifying the values of process attributes.

The PSM completed for the System Test phase of the Incremental Process is given in Fig. 6. Each input, output, activity, role, and tool identified in Fig. 5 was entered into the rows of the PSM once as an expected process attribute. The columns of the matrix indicated individual executions of the System Test phase. Each process attribute in a row was questioned for its existence in every process execution; and if the attribute applied to an execution, it was marked with an "o" in the intersection cell. The completed matrix formed a base to see and assess the consistency of overall executions with respect to the expected process attributes of the System Test phase. It was observed from the matrix that although a number activities were not performed as expected, the executions of the System Test phase were carried out consistently in general.

The step 4 is performed to understand the scope of a software development process with respect to a process reference model like CMMI for Development [10] or ISO/IEC 12207 [46], and is recommended if performances of two or more software development processes will be compared. The execution of this step for a single development process might be beneficial as well for understanding the existence of the activities in the development process with respect to a process reference model. This is a quick mapping of the activities of a software development process to the practices of a process reference model rather than being a formal assessment of the development process. The mapping requires primary knowledge on the practices of the process reference model and on the activities of the development process which are obtained in the steps 2 and 3. First, the processes of the reference model are selected as a base for the mapping by considering the purpose of the analysis and the context of the software development process. Then, the activities of the software development process are



			Features																			
		SYSTEM TEST PHASE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	Inputs																					
	1.1	Feature Requirement Documents 2,3,4	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	1.2	Notes from the Workshop held with the Developer	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
2	Outputs																					
	2.1	System Integration (13) and Regression (12) Test Documents																				
	2.2	Test Results and Status Report	o	o		o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
3	Activities																					
	3.1	Prapare System Integration and Regression Test Documents																				
	3.2	Review the System Integration and Regression Test Documents																				
	3.3	Revise the System Integration and Regression Test Documents																				
	3.4	Prapare System Integration Test Scenarios	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	3.5	Review the System Integration Test Scenarios	o	o	o	o	o	o	o				o	o	o	o			o	o	o	o
	3.6	Revise the System Integration Test Scenarios	o	o	o	o	o	o	o	o			o	o	o	o			o	o	o	o
	3.7	Prepare Test Environment	o	o	o	o	o	o	o		o	o	o	o	o	o	o	o	o	o	o	o
	3.8	Run Quality Assessment Tests	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	3.9	Release the Quality Assessment Test Results and Status Report	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	3.10	Record the Quality Assessment Test Defects	o	o	o	o	o	o	o	o	o	o	o	o	o	o		o	o	o	o	o
	3.11	Remove the Quality Assessment Test Defects	o	o	o	o	o	o	o	o	o	o	o	o	o	o		o	o	o	o	o
	3.12	Retest the Quality Assessment Test Defects	o	o	o	o	o	o	o	o	o	o	o	o	o	o		o	o	o	o	o
	3.13	Run the System Integration Test Scenarios	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	3.14	Release the System Integration Test Results and Status Report	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	3.15	Record the System Integration Test Defects	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	3.16	Remove the System Integration Test Defects	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	3.17	Retest the System Integration Test Defects	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	3.18	Run the Regression Test Scenarios	o	o	o	o			o								o		o		o	
	3.19	Release the Regression Test Results and Status Report	o	o	o	o			o								o		o		o	
	3.20	Record the Regression Test Defects																				
	3.21	Remove the Regression Test Defects																				
	3.22	Retest the Regression Test Defects																				
	3.23	Put the Product into Internal Customer Use	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	3.24	Record the Defects found in Internal Customer Use	o	o	o	o	o	o	o	o		o	o	o	o	o	o	o	o	o	o	o
	3.25	Remove the Defects found in Internal Customer Use	o	o	o	o	o	o	o	o		o	o	o	o	o	o	o	o	o	o	o
	3.26	Retest the Defects found in Internal Customer Use	o	o	o	o	o	o	o	o		o	o	o	o	o	o	o	o	o	o	o
4	Roles																					
	4.1	Project Manager	o	o	o	o	o	o	o	o	o	o	o	o	o	o			o	o	o	o
	4.2	Technical Project Lead	o	o	o	o	o	o	o	o	o	o	o	o	o	o			o	o	o	o
	4.3	Developer (Feature Owner)	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	4.4	Tester	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
5	Tools																					
	5.1	Defect Tracking System	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	5.2	Global Document Management System	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
	5.3	Test Management Tool / File Server	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o

Fig. 6. Process Similarity Matrix (PSM) for System Test phase of the Incremental Process [44].

mapped to the practices of the selected processes of the process reference model by using the activity numbers given in the step 2. The degree of conformance is determined for each process practice in the following four values in the ordinal scale: None, Partial, Large, Full. The degree of the conformance to a process (or a subprocess) is determined by taking the median of the values for the process practices. At the end of this step, the degree of conformance of a software development process to the selected processes of the process reference model is identified. Understanding the scopes of software development processes with respect to the same process reference model prior to comparison is useful in interpreting analysis results and comparing process performances.

For the analysis of the Incremental Process, CMMI for Development was selected as the process reference model and the mapping included the process areas; Requirements Management, Product and Process Quality Assurance, Requirements Development, Technical Solution, Product Integration, Verification, and Validation. Example mapping of the activities of the Incremental Process to the practices of the last process area is given in Table 7. Since each CMMI process areas is structured by specific goals (SG) and by specific practices (SP) under these goals, the conformance to a process area was represented on the basis of the degree of achievement of the specific goals. The degree of achievement for a SG was determined by taking the median of the values in the degree of conformance column filled for the SPs of that SG. For example, the median of the values (P, L, L) in the degree of conformance column filled for the SPs of "SG 1 Prepare for Validation" in Table 7 was L; and, the degree of achievement for SG1 was recorded as "Large". The degrees of achievement for the specific goals of the selected CMMI process areas by the Incremental Process are already available in Table 3.

In the step 5, the analysis goals and required measures are identified by using the GQM Approach [8]. The output of this step is a draft GQM table including analysis goals, related questions, and required measures. Applying the GQM Approach, on the other hand, is not trivial since the approach does not provide much guidance on constructing the hierarchy that it proposes [8]. The application of the previous steps of the analysis method, nevertheless, shows direction in defining the analysis goals by the quantitative knowledge that it elicits on process phases and attributes. Identifying and modeling the components of a software development process provides information on the entities that are subject to measurement. The identification of the questions are related both to the entities to be measured and to software measures themselves. Therefore, the identification and definition of derived and base measures require basic knowledge on software measures, and are not complete until the steps 6, 7, and 8 are carried out.

Regarding the analysis of the Incremental Process, the draft GQM table obtained as the output of the step 5 is not provided here due to the space constraint but its finalized version (as the output of the step 8) is identical to the one shown in Table 4. Since the identified phases of the Incremental Process included Implementation (design and coding), System Test, and Customer Use, and the sequence of these phases made up the entire development process; the goals of analysis were defined as to understand individual performances of Implementation and System Test phases as well as overall performance and product quality of the entire development process. While identifying the questions to satisfy these goals, we utilized the definitions of the most used derived measures such as defect density for the entity of the software product; effectiveness and speed for the entities of the software process and its phases; and effort estimation capability and productivity for the entity of the resource. The formulas of the derived measures defined and used in the evaluation of the Incremental Process are given in Table 8. The base measures in the formulas of the derived measures can be followed from Table 9 with their categories, names, data collection and verification mechanisms, and measure usability attribute ratings and evaluations.

The steps 6, 7, and 8 are recommended to complement the top-down approach of the GQM.

In the step 6, base measures of the software development process are identified by following a bottom-up approach. Process data available on the forms and tools are investigated for each process phase identified in the step 1, and the base measures that might have a relation to the goals identified in the step 5 are determined as candidates for quantitative analysis.

Regarding the analysis of the Incremental Process, process data were collected from tools and forms, and via interviews held with the process performers (Table 9).

In the step 7, process base measures identified in the previous step are evaluated for their usability in quantitative analysis. The Measure Usability Questionnaire described in Section 2.4 is utilized for the evaluation. The output of this step is a list of process measures that are usable in quantitative analysis that will be carried out in relation to the analysis goals.

The list of base measures, whose usability was evaluated as either Strong or Moderate, for the Incremental Process is given in Table 9. The data for the base measures of planned development start date, planned system test start date, planned development finish date, actual development start date, actual system test start date, actual development finish date, and actual system test finish date were not recorded timely and stored properly. Therefore the usability of these measures was evaluated as weak, and they were excluded from the analysis study.

The step 8 is performed to revise the draft GQM table created in the step 5 and to define derived measures that will be used in answering the questions in the table. The goals, questions and measures identified in the draft GQM table are the planned ones rather than being real, and might require update considering the knowledge obtained in the steps 6 and 7. The purpose is to reconcile the required measures identified by using the GQM Approach and the measures identified as usable for quantitative analysis at the end of the step 7. The draft GQM table is revised and finalized. In addition, derived measures, which will be used to answer the questions in the GQM table, are defined with their names and formulas. It is not definite in the GQM Approach how a question will

Table 7
Mapping of Incremental Process activities to the specific practices of CMMI's Validation process area.

Process area	Maturity level	Specific goal	Specific practice	Degree of conformance (N, P, L, F)	Incremental Process Activity No (from System Test phase process model)
Validation	3	SG1. Prepare for Validation	SP1.1. Select Products for Validation	Large (L)	12.c
			SP1.2. Establish the Validation Environment	Large (L)	4.c, 12.c
			SP1.3. Establish Validation Procedures and Criteria	Partial (P)	12.c
		SG2. Validate Product or Product Components	SP2.1. Perform Validation	Large (L)	12.c
			SP2.2. Analyze Validation Results	Large (L)	6.c, 7.c, 8.c, 9.c

Table 8
Derived measures and their formulas used in the evaluation of the Incremental Process.

Goals and Related Derived Measures		Derived Measure Formulas
G1	*Obtain an Understanding of Product Quality*	
D1	Defect density per test scenario in Implementation Phase	B1/B4
D2	Defect density in Implementation Phase	B1/B19
D3	Defect density per test scenario in System Test Phase	B2/B5
D4	Defect density in System Test Phase	B2/B19
D5	Defect density in Customer Use Phase	B3/B19
D6	Overall defect density	(B1 + B2 + B3)/B19
G2	*Obtain an Understanding of System Test Phase Performance*	
D7	System test execution V&V effectiveness	B2/(B1 + B2 + B3)
D8	System test effectiveness	B2/(B10 + B11)
D9	System test speed	B5/(B10 + B11)
D10	System test effort estimation capability	(B10 + B11 + B12 + B13)/B18
G3	*Obtain an Understanding of Development Phase Performance*	
D11	Software productivity	B19/(B6 + B7 + B8 + B9 + B14 + B15 + B16)
D12	Development effort estimation capability	(B6 + B7 + B8 + B9 + B14 + B15 + B16)/B17
D13	Development test execution V&V effectiveness	B1/(B1 + B2 + B3)
D14	Development test effectiveness	B1/(B8 + B9)
D15	Development test speed	B4/(B8 + B9)
G4	*Obtain an Understanding of Incremental Process Performance*	
D16	Incremental Process test execution V&V effectiveness	(B1 + B2)/(B1 + B2 + B3)
D17	Incremental Process defect removal effort ratio	(B14 + B15 + B16)/((B6 + B7 + B8 + B9) + (B10 + B11 + B12 + B13) + (B14 + B15 + B16))
D18	Incremental Process productivity	B19/((B6 + B7 + B8 + B9) + (B10 + B11 + B12 + B13) + (B14 + B15 + B16))
D19	Incremental Process effort estimation capability	((B6 + B7 + B8 + B9) + (B10 + B11 + B12 + B13) + (B14 + B15 + B16))/(B17 + B18)

Table 9
Base measures used in the evaluation of the Incremental Process.

Category	No	Base measure	Data Coll./Ver. Mechanisms	Usability Evaluation by MUQ				
				MUA-1	MUA-2	MUA-3	MUA-4	Evaluation
Number of defects	B1	Number of defects detected in Implementation	Tool/Form	F	F	F	L	Strong
	B2	Number of defects detected in System Test	Tool/Form	F	F	F	L	Strong
	B3	Number of defects detected in Customer Use	Tool	F	F	F	L	Strong
Number of test scenarios	B4	Number of test scenarios run by the developer	Form/Interview	F	F	F	P	Moderate
	B5	Number of test scenarios run by the tester	Tool/Form	F	F	F	L	Strong
Design and implementation effort	B6	Design effort	Form/Interview	F	F	L	P	Moderate
	B7	Coding effort	Form/Interview	F	F	L	P	Moderate
	B8	Test design effort of the developer	Form/Interview	F	F	L	P	Moderate
	B9	Test execution effort of the developer	Form/Interview	F	F	L	P	Moderate
System test effort	B10	Test design effort of the tester	Form/Interview	F	F	F	P	Moderate
	B11	Test execution effort of the tester	Form/Interview	F	F	F	P	Moderate
	B12	Defect reporting effort	Form/Interview	F	F	F	P	Moderate
	B13	Retest effort for detected defects	Form/Interview	F	F	F	P	Moderate
Defect removal effort	B14	Removal effort of defects detected in Implementation	Tool/Interview	F	F	F	P	Moderate
	B15	Removal effort of defects detected in System Test	Tool/Interview	F	F	F	P	Moderate
	B16	Removal effort of defects detected in Customer Use	Tool/Interview	F	F	F	P	Moderate
Planned effort	B17	Planned effort for Design and Implementation	Tool/Interview	F	F	F	L	Strong
	B18	Planned effort for System Test	Tool/Interview	F	F	F	L	Strong
Software length	B19	Source Lines of Code	Tool/Interview	F	F	F	L	Strong

be answered by using the base measures [8], and the formulas of the derived measures indicate how they are obtained from the base measures (Table 8).

Finally, in the step 9, quantitative analysis of the software development process is carried out in accordance to the GQM table and the definitions of the derived measures. The values of the derived measures are calculated from the values of the base measures according to the formulas, the questions are answered by using the values of the derived measures, and the achievement of the goals are evaluated based on the answers to the questions. The results of the quantitative analysis are interpreted by considering the knowledge obtained in the steps 3 and 4 about the scope and context of the software development process. Finally, the results and findings of the quantitative analysis are documented and reported to the interested parties.

5. The results

The analyses and comparison were carried out in a duration of 3 months. First, retrospective analysis of the Incremental Process was completed for the development of 20 features of the product. The effort spent for this analysis was 8 person-days. Second, retrospective analysis of the Agile Process was carried out for the development of another 20 features of the product. The effort of this analysis was 11 person-days. Third, a comparison of the results obtained for the performance and product quality of the Incremental and Agile processes was performed. The effort spent for the comparison and for deriving rationale for the results was 3 person-days.

With the GQM table (Table 4) where common goals were determined for the comparison of different development processes; it was aimed to understand the product quality, the performance of the Implementation phase, the performance of the System Test phase, and the performance of the overall process. Table 10 shows the quantitative results of the comparison. The ratio of the derived measure values in the Agile Process to the values in the Incremental Process are given in the table. A "–" in a cell indicates that the value was either not measured or not applicable.

The performance and the product quality of the processes in common scopes were compared in the following sub-sections by taking the goals and derived measures in the GQM table (Table 4) as the base. In all the comparisons, the values are given after being multiplied by a specific constant in order to keep the secrecy of the company data. In the graphics; phase information is given horizontally, and the average value of the 20 features for the derived measure is given vertically.

5.1. Comparing product quality (G1)

In order to compare the Incremental and Agile processes in terms of product quality, Test Defect Density per Test Scenario (D1, D3) and Test Defect Density per Line of Code (D2, D4, D5, D6) derived measures were taken as the base.

In Fig. 7, the values of the Test Defect Density per Line of Code derived measure of the Incremental and Agile processes are given in comparison in their Implementation, System Test, and Customer phases and in the overall process. Following the figure, it is determined that:

- Test Defect Density in the Implementation phase of the Agile Process is 14% more than that of the Incremental Process.
- Test Defect Density in the System Test phase of the Incremental Process is two times more than that of the Agile Process.

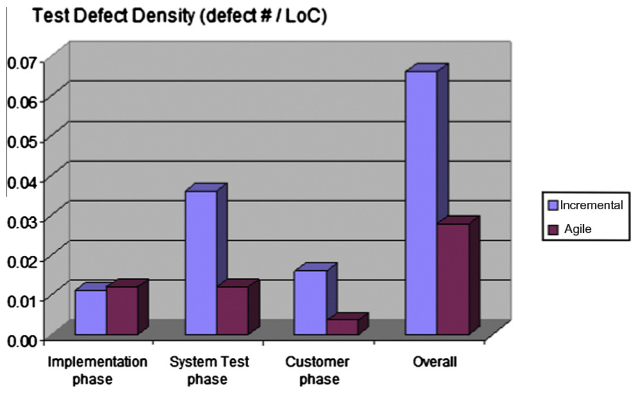


Fig. 7. Test Defect Density (per Line of Code) (D2, D4, D5, D6).

- Test Defect Density in the Customer phase of the Incremental Process is five times more than that of the Agile Process.
- In the overall, the Test Defect Density of the Agile Process is 57% less than that of the Incremental Process.

The reason of this situation was investigated by reviewing and comparing the Process Similarity Matrices completed for each phase of the Incremental and Agile processes. The matrices were obtained from qualitative process enactment data and constituted objective evidence for deriving the rationale behind the quantitative results. It was observed from the matrices that test activities of the Agile Process were practiced in more regular, consistent, automized, and effective way than those of the Incremental Process. For example, in the Implementation phase of the Incremental Process, it was observed that Component and Component Integration Test Documents were generally disregarded, Component Tests were usually not implemented, and Component Integration Tests were disrupted for some features. In addition, Unit Tests were rarely used in the Incremental Process. On the other hand, in the Implementation phase of the Agile Process, it was observed that Unit Tests, Integration Tests, and Smoke Tests were generally employed.

5.2. Comparing the performances of implementation phase, system test phase, and overall process (G2, G3, G4)

In order to compare the performances of the Incremental and Agile processes; Test Execution V&V Effectiveness (D7, D13, D16), Test Effectiveness (D8, D14), Test Speed (D9, D15), Effort Estimation Capability (D10, D12, D19), Productivity (D11, D18), and Defect Removal Effort Ratio (D17) derived measures were taken as the base.

In Fig. 8, the values of the Test Execution V&V Effectiveness derived measure of the Incremental and Agile processes are given in

Table 10
The ratio of derived measure values in the Agile Process to the values in the Incremental Process.

Derived measures	Process			
	Implementation	System test	Customer	Overall
	Agile/Incremental ratio		Agile/Incremental ratio	
Test Defect Density (defect#/TS#)	1.44	0.48	–	–
Test Defect Density (defect#/LoC)	1.14	0.33	0.18	0.43
Test Execution V&V Effectiveness (defect#/defect#)	2.30	0.80	–	1.20
Test Effectiveness (defect#/person-hour)	1.31	0.51	–	–
Test Speed (TS#/person-hour)	0.88	0.96	–	–
Effort Estimation Capability (person-hour/person-hour)	1.00	1.20	–	1.04
Productivity (LoC/person-hour)	1.54	–	–	1.79
Defect Removal Effort Ratio (person-hour/person-hour)	–	–	–	0.76

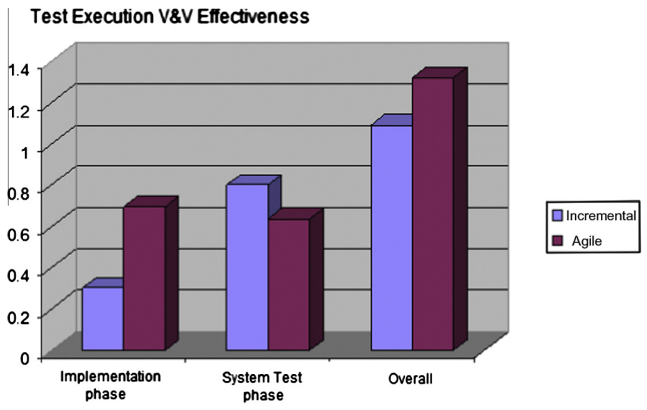


Fig. 8. Test Execution V&V Effectiveness (D7, D13, D16).

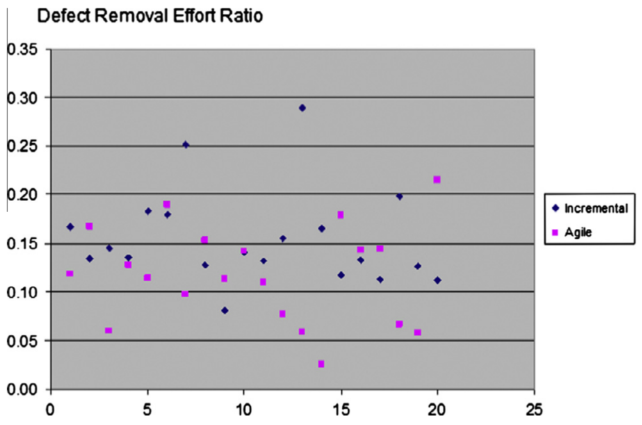


Fig. 9. Defect Removal Effort Ratio (D17).

comparison in their Implementation and System Test phases and in the overall process. Following the figure, it is determined that:

- Test Execution V&V Effectiveness in the Implementation phase of the Agile Process is 2.3 times more than that of the Incremental Process.
- Test Execution V&V Effectiveness in the System Test phase of the Agile Process is 20% less than that of the Incremental Process.
- In the overall, Test Execution V&V Effectiveness in the Agile Process is 20% more than that of the Incremental Process.

The findings that the Test Execution V&V Effectiveness in the Implementation phase is higher and in the System Test phase is lower in the Agile Process is in parallel with the findings of the comparison of the Test Defect Density.

When the values of the Test Effectiveness (D8, D14) derived measure in the Implementation and System Test phases of the Incremental and Agile processes are compared, it is observed that:

- Test Effectiveness in the Implementation phase of the Agile Process is about 30% more than that of the Incremental Process
- Test Effectiveness in the System Test phase of the Agile Process is about the half of that of the Incremental Process.

The comparison of the values of the Test Speed (D19, D15) derived measure in the Implementation and System Test phases of the processes shows that:

- Test Speed in the Implementation phase of the Agile Process is 12% less than that of the Incremental Process.
- Test Speed in the System Test phase of the Agile Process is 4% less than that of the Incremental Process.

When the values of the Effort Estimation Capability (D10, D12, D19) derived measure in the Implementation and System Test phases and in the overall processes are compared, it is determined that:

- Effort Estimation Capability in the Implementation phase of the Agile Process is almost the same as that of the Incremental Process.
- Effort Estimation Capability in the System Test phase of the Agile Process is 20% more than that of the Incremental Process.
- In the overall, Effort Estimation Capability for the Agile Process is 4% more accurate than that of the Incremental Process.

The comparison of the values of the Productivity (D11, D18) derived measure in the Implementation phase and in the overall process indicates that:

- Productivity in the Implementation phase of the Agile Process is 54% more than that of the Incremental Process.
- In the overall, the Agile Process is 79% more productive than the Incremental Process.

Fig. 9 shows the distribution of the values of the Defect Removal Effort Ratio (D17) derived measure in the Incremental and Agile processes to the features. In the figure, the 20 features selected within the scope of the product family are given horizontally, and the values of these features are given vertically.

The distribution of the values in the figure shows that the Agile Process has lower values. Effort Ratio used to remove the defects in the Implementation, System Test and Customer phases of the Agile Process is 24% less than that of the Incremental Process. The rationale for this finding is that the test activities in the Implementation phase of the Incremental process was not as effective as those in the Agile Process and that the defects were detected in later phases with increasing defect removal costs. As it is seen in Fig. 7, Test Defect Density (per Line of Code) in the Customer phase of the Incremental Process is five times more than that of the Agile Process. The later the defects are detected, the more the resolution costs become.

6. Threats to validity

In this section, the validity of the comparison study is evaluated in terms of four tests, which are used commonly to establish the quality of the empirical research, as summarized by Yin [42]. These tests are construct validity, internal validity, external validity, and reliability (or conclusion validity).

The *construct validity* is related to identifying correct operational measures for the concepts being studied [42]. It requires developing a sufficiently operational set of measures and to avoid subjective judgements to collect the data [42]. In the comparison study, the structure of the GQM tables provided the chain of evidence from the measurement data to the achievement of the goals. Widely known measures as defect density, process effectiveness, and productivity were utilized as bases for the comparison. The operational definitions of these measures were made and their usability in the analyses were evaluated by using Measure Usability Questionnaires. Data belonging to the base measures was collected by tools and forms, and verified by holding interviews with data providers and by cross-checking with produced documents where available.

The *internal validity* is applicable for explanatory or causal studies only and not for descriptive or exploratory studies [42]. It requires seeking to establish a causal relationship, whereby certain conditions are believed to lead to other conditions, as distinguished from spurious relationships [42]. Threats to internal validity for our study include the degree of similarity of the context in the analyses, the subjectivity in the interpretation of the qualitative methods used, and the existence of domain knowledge in the project team during the Agile Process.

The domain of the developments was the same, and the average sizes of the selected features in the projects were approximate though the features themselves were not identical. The infrastructure, the programming language, and the development environment were the same, and the number of persons employed in common in the projects was 7 (out of 10 in the project A). The amount of the features selected for the analyses and comparison was 20, which is the minimum requirement for quantitative results to be representative [43]. However, personal skills of team members were not taken into consideration in the study, which is an internal threat to the validity. The scope of the comparison included software development and quality assurance processes, but not project management processes. The exclusion of the project management practices in comparison is another threat to the internal validity. These practices might have had consequences on the performance of the development and testing processes but our observation is that their inclusion would not create a significant difference in the scopes compared since both projects applied basic project management practices. Matching of the scopes of the Incremental and Agile processes to the selected process areas of the CMMI for Development in the ordinal scale assured the equality of process scopes for the comparison. However, the matching of the process activities was performed to the specific practices, but not to the generic practices, of the selected process areas.

Qualitative methods included matching of the process scopes, measure usability evaluation, and process similarity analysis. The subjectivity in using these methods was minimized by the utilization of appropriate assets such as Process Similarity Matrix and Measure Usability Questionnaire, and by determining the rules on how to interpret their outputs. Although using these assets might still have introduced some subjectivity into the analysis; their adoption in the study is considered beneficial rather than detrimental. For example, using Process Similarity Matrices in the data collection stage enabled easy elicitation of the rationale behind the results of the comparison in the composition stage.

The developments of 20 features were selected for the analyses of the Incremental and Agile processes, and the team members were mostly the same in both developments. In other words, most team members had the domain knowledge and the experience with the development environment while carrying out the Agile process. This situation might have increased the performance and the product quality of the Agile process. In our case we could not find a way to measure the effect of this situation on the comparison results, and accept such a threat to the internal validity. A solution might be the development of the features by an independent team of similar size and skills by following the Agile process. In the study, this was not possible due to the lack of human resources.

The *external validity* is about defining the domain to which a study's findings can be generalized [42]. Threats to external validity for our study include the generalizability of the comparison results and the usability of the analysis method by other experts.

For the comparison of the Incremental Process and Agile Process, this is the only study made that is based on the development of the same product in the company. The results of the comparison showed that the Agile Process has performed better than the Incremental Process for this development, but it is not definite if this result will be the same in other product developments in the company or in the developments of other companies in a broader context. Therefore the conclusions derived by this study are sound only in itself and cannot be generalized. We need to conduct further case studies to investigate the generalizability of the results.

The purpose of the comparison study was not to evaluate general usability of the analysis method, and therefore the usability of the method by other experts was not questioned in this study. We need to carry out case studies with this specific aim, prior to testing the method's usefulness in broader contexts.

As the last test, the *reliability* is related to demonstrating that the operations of a study can be repeated, with the same results. Here the emphasis is on doing the same case over again, not on replicating the results of one case by doing another case study – which is vital to external validity [42]. In our study, the analysis method defined in Section 4 was used to analyse the performance and product quality of both the Incremental and Agile processes, and therefore enabled the methodological replication of the analyses.

7. Conclusion

Software management should rely on objective evidence rather than subjective judgements to have better project success rates. The decisions of changing a key technology, adapting a life-cycle model, or shipping a software product require careful investigation of the current situation and the likely consequences. This investigation is possible with objective analyses on qualitative and quantitative data characterizing the phenomena. Unfortunately, the integrated use of qualitative and quantitative methods is not very common in software engineering.

In this paper, we described an empirical method that proposes the utilization of qualitative and quantitative methods together to systematically analyze and compare the performances of more than one software development process and to derive conclusions and the rationale. The analysis method was constructed as a practical guideline for the retrospective analyses and comparison of the Incremental and Agile processes employed in two different projects of a middle-size, telecommunication software company. The study was completed in a duration of 3 months and took an effort of 22 person-days. The definition of the empirical method enabled effective and efficient replication of the analyses for both processes.

The results of the comparison demonstrated that the Agile Process had performed better than the Incremental Process in terms of productivity (79%), defect density (57%), defect resolution effort ratio (26%), Test Execution V&V Effectiveness (21%), and effort prediction capability (4%). These results indicated that development performance and product quality achieved by following the Agile Process was superior to those achieved by following the Incremental Process in the projects compared.

The acts of measurement and analysis almost always result in understanding and improvement in the context that they are applied. This study is not an exception. The utilization of the GQM Approach, in addition to providing practicality in identifying analysis goals and related questions and measures, supported orderly evaluation and understanding of the results. The matching of the development processes to CMMI for Development enabled to realize the issues for improvement to achieve CMMI maturity level 3. The assessment of process similarity helped to observe the activities that were disregarded in the executions of the development processes. Evaluating the measures' usability for quantitative analysis made it possible to identify improvements on the measures and their data collection mechanisms. As a whole, the study enabled a comprehensive review of the two development processes and their strengths and weaknesses, and constituted objective evidence for organization-wide deployment of the Agile Process in the company.

In our study we could not have the chance of following the long-term effects of the transition from the Incremental Process to the Agile Process within the company, which would provide a more complete view. Investigating the outputs of the developments in terms of the entities of product, process, and resource before and after such a transition would be interesting subject to further research. As the future work, we aim more case studies to test the usefulness of the analysis method in broader contexts and to understand the usability of the method by independent analysers.

References

[1] I. Sommerville, Software Engineering, ninth ed., Addison-Wesley, 2010.
[2] Internet: "The Standish Group". <http://www.standishgroup.com/>.
[3] Internet: "Manifesto for Agile Software Dev." <http://AgileManifesto.org>.
[4] T. Dyba, T. Dingsøyr, Empirical studies of agile software development: a systematic review, Inform. Softw. Technol. 50 (9–10) (2008) 833–859.
[5] K. Petersen, C. Wohlin, The effect of moving from a plan-driven to an incremental and agile development approach: an industrial case study, Emp. Softw. Eng. 15 (6) (2010) 654–693.
[6] G. Van Waardenburg, Hans. van Vliet, When agile meets the enterprise, Inform. Softw. Technol. 55 (12) (2013) 2154–2171.
[7] O. Gómez, H. Oktaba, M. Piattini, F. García, A systematic review measurement in software engineering: state-of-the-art in measures, ICSOFT 2006, CCIS 10, LNCS 5007, 2008, pp. 165–176.
[8] R. Van Solingen, V. Basili, G. Caldiera, D.H. Rombach, Goal Question Metric (GQM) Approach, in: Encyclopedia of Software Engineering, online vers.@Wiley Interscience, John Wiley & Sons, 2002.
[9] ISO, ISO/IEC 15939: Software Measurement Process, 2007.
[10] CMU/SEI-CMMI Product Team, CMMI for, Development V1.3, CMU/SEI-2010-TR-033, 2010.
[11] A. Tarhan, O. Demirors, Apply quantitative management now, IEEE Softw. 29 (3) (2012) 77–85.
[12] P. Abrahamson, O. Salo, J. Ronkainen, J. Warsta, Agile software development methods: review and analysis, VTT, 2002.
[13] A.A. Puntambekar, Software Engineering, Technical Publications, 2009.
[14] B.W. Boehm, A spiral model of software development and enhancement, IEEE Computer 21 (1988) 61–67.
[15] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, 2004.
[16] S.R. Palmer, J.M. Felsing, A Practical Guide to Feature-Driven Development, 2002.
[17] A. Cockburn, Agile Software Development, Addison-Wesley, 2002.
[18] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2001.
[19] D.J. Anderson, Agile Management for Software Engineering, Prentice-Hall, 2008.
[20] J. Highsmith, Agile Software Development Ecosystems, Addison Wesley, 2002.
[21] M. Poppendieck, T. Poppendieck, Implementing Lean Software Development, Addison-Wesley, 2006.
[22] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley Object Tech Series, 1999.
[23] N.E. Fenton, J. Bieman, Software Metrics: A Rigorous and Practical Approach, third ed., CRC Press, 2013.
[24] C. Ebert, R. Dumke, Software Measurement: Establish – Extract – Evaluate – Execute, Springer, 2007.
[25] IEEE Std 1012 – IEEE Standard for Software Verification and Validation, 2004.
[26] A. Tarhan, O. Demirors, Investigating the effect of variations in test development process: a case from a safety-critical system, Softw. Qual. J. 19 (4) (2011) 615–642.
[27] D.J. Wheeler, Advanced Topics in Statistical Process Control, SPC Press, 1995.
[28] W.A. Florac, A.D. Carleton, Measuring the Software Process: Statistical Process Control for Software Process Improvement, Pearson Education, 1999.
[29] Y. Khramov, The cost of code quality, in: Proceedings of the AGILE Conference, 2006, pp. 119–125.
[30] G. Benefield, Rolling out Agile in a large enterprise, in: Proceedings of the 41st Annual Hawaii International Conference on System Sciences, IEEE Computer Society, 2008, 461.
[31] F. Ji, T. Sedano, Comparing extreme programming and Waterfall project results, Software Engineering Education and Training (CSEE&T), in: Proceedings of 24th IEEE-CS Conference, 22–24 May 2011, pp. 482–486.
[32] F. Macias, M. Holcombe, M. Gheorghe, A formal experiment comparing extreme programming with traditional software construction, in: Proceedings of the Fourth Mexican International Conference on Computer Science (ENC), 2003, pp. 73–80.
[33] A. Dagnino, K. Smiley, H. Srikanth, A.I. Anton, L. Williams, Experiences in applying agile software development practices in new product development, in: Proceedings of the 8th International Conference on Software Engineering and Applications (IASTED), 2004, pp. 501–506.
[34] S. Ilieva, P. Ivanov, E. Stefanova, Analyses of an agile methodology implementation, in: Proceedings of the 30th Euromicro Conference, IEEE Computer Society, 2004, 326–333.
[35] L. Layman, L. Williams, L. Cunningham, Exploring extreme programming in context: an industrial case study, in: Proceedings of the Agile Development Conference, 2004, pp. 32–41.
[36] D. Dalcher, O. Benediktsson, H. Thorbergsson, Development life cycle management: a multiproject experiment, in: Proceedings of the 12th International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05), 2005, pp. 289–296.
[37] A. Sillitti, M. Ceschi, B. Russo, G. Succi, Managing uncertainty in requirements: a survey in documentation-driven and agile companies, in: Proceedings of the 11th International Software Metrics Symposium (METRICS), 2005, pp. 10–17.
[38] C.A. Wellington, T. Briggs, C.D. Girard, Comparison of student experiences with plan-driven and agile methodologies, in: Proceedings of the 35th ASEE/IEEE Frontiers in Education Conference, 2005, pp. T3G/18-23.
[39] P.J. Rundle, R.G. Dewar, Using return on investment to compare agile and plan-driven practices in undergraduate group projects, in: Proceeding of: 28th International Conference on Software Engineering, 2006, pp. 649–654.
[40] J. Sutherland, C.R. Jakobsen, K. Johnson, Scrum and CMMI Level 5: The Magic Potion for Code Warriors, in: Proceedings of the AGILE Conference, IEEE Computer Society, 2007, pp. 272–278.
[41] J. Li, N.M. Moe, T. Dyba, Transition from a plan-driven process to scrum-a longitudinal case study on software quality, in: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, 2010.
[42] R.K. Yin, Case Study Research: Design and Methods, Applied Social Research Methods Series, vol. 5, SAGE Publications, 2003.
[43] A. Burr, M. Owen, Statistical Methods for Software Quality, International Thomson Computer Press, 1996.
[44] S.G. Yilmaz, A. Tarhan, Systematic analysis of the incremental process as a base for comparison with the agile process, in: Proceedings of the Evaluation and Assessment in Software Engineering (EASE) Conference, 2011, pp. 86–91.
[45] B.J. Hommes, The Evaluation of Business Process Modeling Techniques, TU Delft, 2004. p. 137.
[46] ISO/IEC, ISO/IEC 12207: Systems and Software Engineering-Software Life Cycle Processes, 2008.